VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JAVASCRIPTOVÉ KNIHOVNY A JEJICH VYUŽITÍ PŘI TVORBĚ INTERAKTIVNÍCH WEBOVÝCH APLIKACÍ

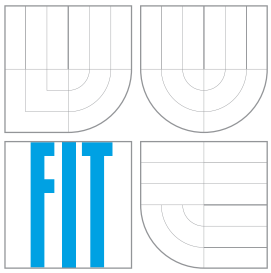
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ WAGNER

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

JAVASCRIPTOVÉ KNIHOVNY A JEJICH VYUŽITÍ PŘI TVORBĚ INTERAKTIVNÍCH WEBOVÝCH APLIKACÍ

JAVASCRIPT LIBRARIES AND THEIR APPLICATION IN BUILDING INTERACTIVE WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ WAGNER
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ MÁČEL
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá javascriptovými knihovnami a jejich využitím při tvorbě interaktivních webových aplikací. Popisuje rovněž technologie potřebné při tvorbě interaktivní webové aplikace. Pomocí javascriptových knihopen Dojo, jQuery UI a Qooxdoo jsou implementovány tři prototypy webového rozhraní emailového klienta. Na základě jejich porovnání jsou demostrovány možnosti využití zvolených knihoven.

Abstract

The bachelor's thesis explores the broad use of JavaScript libraries in the area of interactive web applications. It also describes the base technologies upon which these libraries are build. The thesis consists of model implementations using some of the available libraries namely Dojo, jQuery UI and Qooxdoo each used to create an email client application. The resulting solutions are compared in terms of their diverse qualities.

Klíčová slova

AJAX, DOM, Dojo, JavaScript, jQuery UI, Qooxdoo, RIA

Keywords

AJAX, DOM, Dojo, JavaScript, jQuery UI, Qooxdoo, RIA

Citace

Tomáš Wagner: Javascriptové knihovny a jejich využití při tvorbě interaktivních webových aplikací, bakalářská práce, Brno, FIT VUT v Brně, 2010

Javascriptové knihovny a jejich využití při tvorbě interaktivních webových aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Máčela. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Wagner
19. května 2010

Poděkování

Rád bych poděkoval panu Ing. Máčelovi, za pomocné korekce a rady. Dále bych chtěl poděkovat Dorotě za morální podporu.

Obsah

Kapitola 1

Úvod

S rozvojem internetu a webových služeb, které poskytuje, se možnosti jeho využití postupně mění. Od statických dokumentů, které byly vytvářeny tak, aby podávaly co nejvyšší informační hodnotu, se přechází až k interaktivním webovým aplikacím. Tyto aplikace značně rozšiřují možnosti využití webu. Nové webové technologie dovolují spojit funkcionalitu typickou pro desktopové aplikace s výhodami, které poskytuje internetové médium.

Hlavním cílem práce je seznámit čtenáře s prostředky pro tvorbu interaktivních webových prezentací. Při vývoji interaktivních webových aplikací je na výběr z několika technologií, jakými jsou například JavaScript, Adobe Flex, Flash, Java apod. Tato práce se zaměřuje na oblast klientských technologií a to konkrétně oblast JavaScriptu a již implementovaných knihoven. Prostřednictvím třech vybraných knihoven (Dojo, jQuery UI a Qooxdoo) je proveden návrh implementace emailového klienta. Na těchto příkladech je demonstrováno maximální využití knihoven a nabízených komponent pro tvorbu grafického uživatelského rozhraní. Nejdůležitější částí práce je konečné srovnání z hlediska složitosti implementace, rychlosti vytváření webového rozhraní a práce s uživatelským rozhraním.

V kapitole 2 jsem uvedu definici interaktivní webové aplikace. Zaměřím se na její typické charaketristiky a zavedu pojem RIA. Na závěr je uvedeno několik příkladů, které demonstrují současné využití interaktivních webových aplikací.

V kapitole 3 zmíním jaké prostředky lze v současné době využít pro tvorbu webového rozhraní. V první části popíši jazyky HTML, JavaScript a CSS. V podkapitole o jazyce JavaScript se dále zaměřím na samotnou definici jazyka, na jeho historický vývoj a na podporu a rychlost zpracování v jednotlivých webových prohlížečích. Dále se zaměřím na rozhraní pro práci s objektovým modelem dokumentu - DOM. Specifikuji atributy, metody, proces šíření a zpracování událostí. Poté zmíním techniku, která se označuje AJAX a která dovoluje komunikaci na základě asynchronních požadavků na server. Na závěr zmíním knihovny jazyka JavaScript, jejich výhody a vhodnost použití.

V kapitole 4 představím jednotlivé knihovny, které jsem si vybral pro implementaci a závěrečné srovnání. Zaměřím se přitom na funkce, které každá knihovna poskytuje. Dále pak zmíním rozdíly, v jakých se jednotlivé knihovny od sebe liší. Jak se liší způsob práce s konkrétní knihovnou apod.

V kapitole 5 provedu popis návrhu emailového klienta. Popis zahrnuje kompletní návrh grafického uživatelského rozhraní rozděleného do konkrétních sekcí. Dále je zde naznačená funkčnost, kterou by měly aplikace podporovat, případně dialogová okna, která budou využita ve výsledné aplikaci.

V kapitole 6 popíši implementované klienty, které byly vytvořeny v jednotlivých knihovnách. Jednotlivě popíši využití dané knihovny přímo na příkladech ze vzorové implementace.

V kapitole 7 provedu závěrečné srovnání knihoven dle vybraných hledisek. Např. obsluha událostí, nabídka komponent GUI, manipulace s daty atd.

V kapitole 8 jsou zhodnoceny dosažené výsledky práce. Je provedeno celkové zhodnocení, přínos práce a pohled na možnosti rozšíření.

1.1 Značení v textu

V textu bude použito tučného zvýraznění písma u důležitých tvrzení. Příklady zdrojových kódů v textu budou napsány neproporcionálním písmem.

Kapitola 2

Interaktivní webová aplikace

V současné době převažuje využití webových technologií, jako sady dokumentů, které jsou vzájemně provázány. Jedná se o dokumenty, jejichž obsah definoval sám autor a čtenáři dokumentu je předložen v neměnné (statické) podobě. Samotný čtenář si pak pouze vybírá jaký z dokumentů chce mít dostupný na svém zobrazovacím zařízení. Umožňuje mu to bezstavový HTTP (Hypertext Transfer Protocol) protokol, prostřednictvím kterého uživatel zasílá požadavky a získává odpovědi ve formě hypertextového dokumentu.

Dnes již existuje nový směr a to ve formě interaktivní webové aplikace. Interaktivní webovou aplikaci lze popsat jako webový dokument s přidanou interaktivitou, která zahrnuje reakce na uživatelské vstupy. Hlavním problémem při vytváření webových aplikací byla počáteční definice webového prostoru, jako statických dokumentů. Jelikož interaktivní webové aplikace vyžadují možnost změn v stávajícím dokumentu tak, aby utvářely funkčnost pro kterou byly vytvořené. Za pomocí bezstavového komunikačního protokolu HTTP nebylo možné dosáhnout patřičné funkčnosti. Uchování stavu, který je pro všechny aplikace nutný nebylo možné.

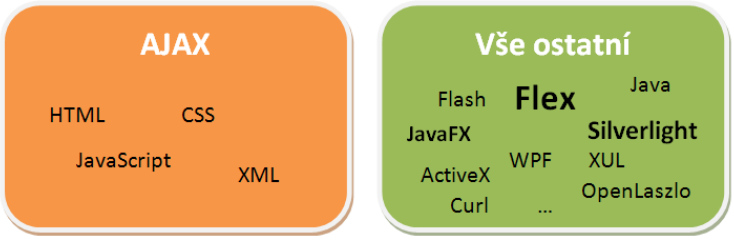
Mezi výhody interaktivních aplikací lze zařadit fakt, že používání je pro uživatele velice jednoduché. Velká spousta takovýchto aplikací je vytvořena tak, aby jejich ovládání bylo na první pohled intuitivní a pohodlné. Běžnému uživateli, který využívá počítač jako prostředek pro získávání informací většinou nečiní používání problémy. Prostředí, ve kterém se nachází, jakožto prostředí webového prohlížeče je pro spoustu uživatelů synonymem pro jednoduché, pohodlné a známé ovládání. Uživatel má webové prostředí také spojené s tím, že se nemusí učit nic nového a že dostane to, co chce, dostupným, rychlým a snadným způsobem.

Interaktivnost u webové aplikace je schopnost aplikace komunikovat a reagovat na řadu vstupů a akcí uživatele. Aplikace by měla být schopna reagovat na vstupy z klávesnice a myši. Mezi vstupy z klávesnice lze zařadit nejen stiskutí určité klávesy, ale také možnost vstupu jako kombinace více kláves tzv. klávesové zkratky. Z myši lze sledovat vstupy, jako kliknutí, pohyb myši nad příslušným objektem, aktuální pozici myši. Aplikace poté tyto vstupy sleduje, dle příslušného vstupu vyhodnocuje a provádí vytváření příslušných výstupů. Interaktivní webová aplikace umožňuje získávání nových informací prostřednictvím asynchronní komunikace se serverem. Synonymem pro interaktivní webové aplikace se stalo označení RIA, které definuje typické rysy.

2.1 RIA

RIA (Rich Internet Application) je označení pro webové aplikace [23]. Kombinují vlastnosti aplikací spuštěných lokálně v operačním systému známých z desktopového prostředí s možnostmi, které nabízí webové prostředí. Desktopové prostředí zachovává formu aplikace, která je pro uživatele osobního počítače snadno dostupná, snadno ovladatelná, rychlá a konfigurovatelná.

Mezi charakteristické vlastnosti „bohatých“ interaktivních webových aplikací patří uživatelské rozhraní, které kopíruje vlastnosti aplikací z desktopového prostředí. V ovládání může být využito například klávesových zkratek, ovládání rolovacím kolečkem myši, přesunováním různých objektů. Reakcí na uživatelské akce se pak vytváří určitá požadovaná funkcionalita. Webová aplikace již tedy není statická stránka, jejímž cílem je podat uživateli potřebné informace. V tomto případě se jedná dokumenty s přidanou interaktivitou na straně uživatele.



Obrázek 2.1: Oblasti aplikací označených jako RIA.Převzato z [20]

Dle obrázku 2.1 lze RIA aplikace teoreticky rozdělit do dvou skupin: První skupina označená jako AJAX (Asynchronous JavaScript and XML), obsahuje aplikace napsané v jazyce JavaScript uplatňující asynchronní komunikaci se serverem. JavaScript je jazykem interpretovaným na straně klienta. Většina dnes využívaných prohlížečů již interprety jazyka JavaScript obsahují. Tento druh aplikací nevyžaduje přídavnou instalaci doplňků do webového prohlížeče. Proto je právě v této skupině kladen důraz zejména na funkčnost v co nejširším spektru webových prohlížečů.

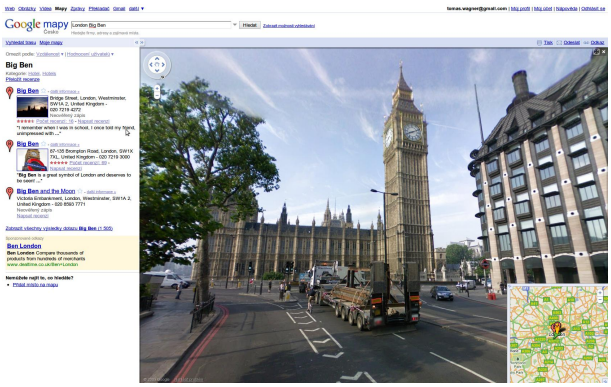
Druhá skupina o které se zmíním, obsahuje technologie, jako je například Microsoft Silverlight, Adobe Flash, Shockwave nebo Flex. Tyto technologie není možné spustit ve standartním webovém prohlížeči. Pro jejich běh je potřebné do prohlížeče doinstalovat příslušný zásuvný modul (plugin) od dané firmy. Modul musí být schopen technologii ve webové stránce detekovat a umožnit uživateli aplikaci (script) využívat. Takový přístup může být vhodný zejmena v komplikovanějších uživatelských rozhraních, případně pro určité konkrétní aplikace jako je přehrávání multimédií apod.

2.1.1 Příklady reálných interaktivních webových aplikací

Interaktivní webové aplikace jsou již v oblasti internetu značně rozšířené. Může se jednat o jednodušší aplikace až o velmi složité a propracované celky. Pro představu o RIA aplikacích jsem vybral několik názorných příkladů. Jejich využití je například v sociálních sítích služeb (Twitter, Facebook ..), sofistikovaných aplikací pro zobrazování map (Seznam mapy, Google Maps ..) nebo v desktopových prostředích, které částečně simulují pracovní plochu v operáčním systému (Netvibes).

Google Maps

Aplikace společnosti Google s názvem Google Maps umožňuje procházení map celého světa. Obsahuje klasické, satelitní a terénní mapy. Zajímavou část aplikace tvoří tzv. Street View. Uživatel má možnost procházet fotografiemi, které jsou uspořádány tak, aby tvořily kompletní prostor, jako kdyby se člověk nacházel přímo na daném místě. Jedná se o typycký příklad RIA aplikace, kde můžeme najít například ovládání pohybu v prostoru za pomocí šipek klávesnice. Pohyb se uživateli jeví jako realný pohyb v prostoru. Dle polohy v jaké se uživatel na mapě nachází mu jsou nabídnuty různé funkce, jako například zobrazení fotografie nebo videa přímo z daného místa. Aplikace je vytvořena pouze za použití JavaScriptu a asynchronní komunikace se serverem za účelem získávání nových dat například při změně polohy nebo pohybu v daném prostoru.



Obrázek 2.2: Ukázka z aplikace Google Maps při procházení Londýna za pomocí Street View

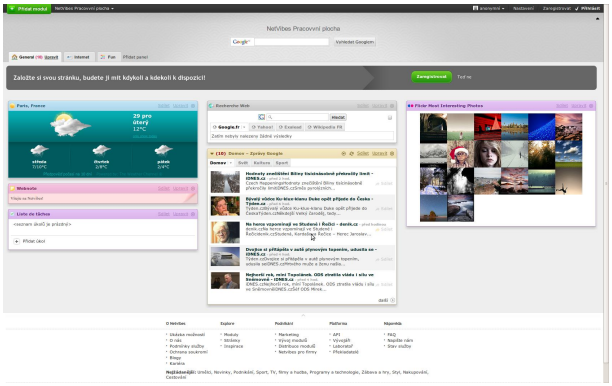
Uživateli tedy k prohlížení map a ulic stačí pouze osobní počítač, běžný libovolný internetový prohlížeč s interpretem jazyka JavaScript. Díky této přístupnosti získala aplikace Google Maps velké popularity. Google byl také jeden z prvních, který prosadil možnosti interaktivních webových aplikací do běžného statického internetu [25].

Netvibes

Interaktivní webové aplikace ovlivnily také oblast tzv. webových desktopových prostředí. Tato desktopová prostředí mohou být pro řadu uživatelů přínosná. Hlavní nápad spočívá v centralizovaném uložení všech dat jako v lokálním počítači, ovšem s tou výhodou, že přístup k datům je uživateli umožněn z libovolného místa na světě, pouze za použití webového prohlížeče. Aplikace Netvibes poskytuje registrovanému uživateli vytvořit si vlastní pracovní plochu a pracovat s ní prostřednictvím libovolného internetového prohlížeče s interpretem jazyka JavaScript. Po přihlášení je uživateli nabídnuta možnost přidávat a spravovat jednotlivé panely. Uživatel si může nastavit, jaký modul má být právě zobrazen na jeho pracovní ploše. Lze si přitom vybírat z modulů jako počasí, zprávy různých zpravodajských serverů, přidávání poznámek, mapy, fotogalerie apod.

Facebook

Aplikace Facebook se řadí mezi velké množství sociálních sítí proslavených zejména ve Spojených státech amerických. Jedná se o internetovou aplikaci pro vzájemnou komunikaci



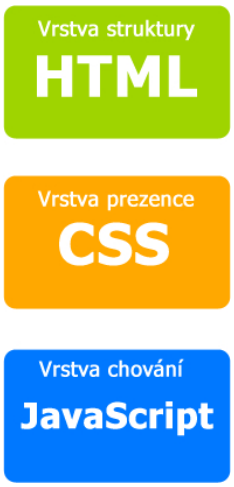
Obrázek 2.3: Plocha aplikace Netvibes

a sdílení informací mezi uživateli. Tato aplikace umožňuje uživateli vytvořit svůj profil a definovat ostatní uživatele, kteří budou moci sledovat činnost daného uživatele. Uživatelé mohou do aplikace psát informace o tom co dělají, sdílet fotogalerie, videa a internetové odkazy. Aplikace Facebook je vytvořena z velké části za pomocí jazyka JavaScript a asynchronní komunikační technologie. Oblíbenost aplikace rapidně vzrostla a v současné době tato sociální síť zahrnuje 400 miliónů aktivních uživatelů [11]. Z tohoto pohledu je zřejmé, že využití interaktivních aplikací je možné i v takto velkých projektech pro obrovské množství uživatelů.

Kapitola 3

Prostředky pro tvorbu dynamických webových aplikací

V této kapitole se konkrétněji zaměřím na jednotlivé vrstvy třívrstvého modelu moderní webové aplikace, tak jak jsem ho vyznačil na obr.3.1 [22, s.36]. Základem je vrstrva struktury, což je textový dokument jazyka HTML. Je definována samotná struktura obsahu a sémantika dokumentu. Druhou vrstvou je vrstva prezenční tvořená CSS. Popisuje vzhled dokumentu, tak jak bude prezentován uživateli. Jedná se o formátování nadpisů, textu apod. Poslední vrstrvou je vrstva chování. Ta je reprezentována jazykem JavaScript, který umožňuje vytvářet reakce na chování uživatele. Prostřednictvím DOM rozhraní, které je dostupné z jazyka JavaScript interperetovaného webovým prohlížečem je pak možnost přistupovat k jednotlivým prvkům dokumentu HTML.



Obrázek 3.1: Třívrstvý model moderní webové aplikace

3.1 HTML a CSS

HTML (HyperText Markup Language) patří do rodiny značkovacích jazyků SGML (Standart Generalized Markup Language). Jazyk HTML je definován množinou prvků. Každý prvek má příslušný sémantický význam. Jednotlivé prvky mají určitou množinu atributů, které blíže specifikují vlastnosti daného prvku. Dokument je tvořen nadpisy, odstavci,

seznamy, obrázky a multimédii. Je definována také jednoduché propojení dokumentů skrze hypertextové odkazy.

HTML dokument se skládá z:

- reference na definici typu dokumentu (DTD) - prvek `doctype`,
- hlavičky dokumentu - prvek `head`,
- těla dokumentu - prvek `body`.

Definice typu dokumentu DTD je kolekce deklarací, která určujuje povolenou strukturu prvků a jejich atributů, které se mohou použít v dokumentu [2]. Dále určuje jaká verze HTML byla použita v daném dokumentu. V současné době se využívá DTD HTML 4.01 a XHTML 1.0 Strict. Dle DTD je možné provádět syntaktickou kontrolu dokumentu a zjišťovat, zda daný dokument dodržuje standardy definované W3C konsorciem.

V hlavičce dokumentu je možné uvést název dokumentu - prvek `title` a umožnit propojení se soubory, které jsou nutné pro správnou reprezentaci daného dokumentu. Může se jednat například o reference na soubory s kaskádovými styly nebo skripty jazyka JavaScript. Je nutné také zahrnout prvky typu `meta`, které určují například typ obsahu dokumentu a kódování.

Prvek `body` obsahuje různe další prvky, které formují strukturu dokumentu. Zmíním například prvky jako `div` a `span`, které vymezují blokové a řádkové oblasti v dokumentu. Dále jsou zde prvky jako `table`, prvky pro tvorbu formulářů jako `input`, `select` pro tvorbu rolovacího menu a prvek `p` pro formátování textu do odstavců. Důležitým prvkem je prvek `a`, který definuje hypertextový odkaz s jehož využitím lze jednotlivé dokumenty propojovat a umožnit tak uživateli přístup do jiného dokumentu. Kompletní výčet prvků značkovacího jazyka HTML4 je možné najít v [1].

CSS (Cascading Style Sheets) - jedná se o kolekci metod tzv. kaskádových stylů, které lze použít ke změně grafické reprezentace a formátování dokumentů HTML. Jedním z cílů kaskádových stylů bylo striktně oddělit obsah dokumentu a definici vzhledu. To umožňuje rychlejší a efektivnější návrhy formátování a také případnou znovupoužitelnost. Kaskádový styl obsahuje pravidla, která se skládají ze selektoru a deklarace hodnoty jedné nebo více grafických vlastností. Využití kaskádových stylů se vždy váže za pomocí selektoru na množinu prvků, která byla tímto selektorem vybrána. Selektory lze tedy cíleně vybrat uržitý konkrétní prvek, případně množinu prvků, které mají společné atributy.

Na každý selektor je možné aplikovat množinu pravidel. Mezi základní pravidla uvedu například pravidla pro formátování textu, jako je změna velikosti textu, fontu a barvy. Dále pak pravidla, která mohou měnit vlastnosti prvků, které vytváří rozložení dokumentu. U těchto prvků je možné nastavit pevnou, nebo relativní velikost a umístění prvku, vnitřní nebo vnější odsazení, barevné orámování a vyplnění. Existují zde také pravidla, která umožňují definovat prostorovou osu „z“ kartézské soustavy souřadnic. Nastavením této osy je možné využít například při překrývání bloků, kde je možné nastavit, v jakém pořadí a které bloky se budou překrývat.

V kaskádových stylech je definována dědičnost. Pokud není dáno jinak, vlastnosti rodičovského prvku v dokumentu přebírají i všechny objekty uvnitř tohoto prvku. Nedědí se ale všechny hodnoty, každá vlastnost je přesně specifikována zda je dědičná. Kompletní specifikaci kaskádových stylů CSS je možné naleznout [5].

3.2 JavaScript

JavaScript je multiplatformním interpretovaným objektově orientovaným programovacím jazykem. Programový kód napsaný v jazyce JavaScript je klientský skript, a proto všechny programy napsané v tomto jazyce jsou interpretovány na klientské stanici v konkrétním webovém prohlížeči. Klientský skript, který je ve zdrojovém kódu, se vkládá do html už při načtení stránky nebo případně i za běhu. Syntaktická podoba JavaScriptu se podobá syntaxi jazyka C avšak využívá dynamického typování.

V roce 1995 byl JavaScript představen firmou Netscape na společné konferenci. Tvůrcem jazyka je Brandon Etch. JavaScript byl standardizován organizací ECMA (European Computer Manufacturers Association) jako ECMAScript. [27, s.46] V této době přišel Microsoft s vlastní implementací nazvanou JScript. Ačkoli JScript vypadal syntakticky stejně jako JavaScript, tak některé JavaScriptové programy nefungovaly v prohlížeči Internet Explorer využívající jako interpret JScript a naopak. Ke standardizaci v jednotlivých prohlížečích docházelo postupně a proto se podpora v jednotlivých prohlížečích liší. Největší rozdíly jsou v API, přístupu k DOM a komunikaci (objekt XMLHttpRequest). Více o JavaScriptu se lze dozvědět z [24] a [21].

3.2.1 Podpora ze strany prohlížečů

Dnes je technologie JavaScriptu podporována v celé řadě klientských webových prohlížečů jako je Internet Explorer, Safari, Chrome, Firefox či Opera a dalších. V jádrech interpretů je však snaha dodržovat současný standard ECMAScript v 3.0.

Internet Explorer V současné verzi prohlížeče Internet Explorer 8 je využito JScript verze 5.8. Tato verze JScriptu je již plně ekvivalentní s JavaScript 1.5. O samotné implementaci tohoto interpertu se moc neví. Jelikož se jedná o komerční produkt společnosti Microsoft, zdrojový kód nebyl nikdy zveřejněn.

Safari Společnost Apple stojí za vývojem prohlížeče Safari a jeho enginem SquirrelFish. Dokáže kompilovat JavaScript do byte kódu, který je poté interpretován prostřednictvím virtuálního stroje. SquirrelFish je napsán v jazyce C++ a šířen jako opensource.

Google Chrome Chrome, prohlížeč společnosti Google využívající engine V8. Engine je napsán v jazyce C++. Kompilací JavaScriptu do nativního strojového kódu před spuštěním se mnohonásobně zvýšila rychlost interpretace kódu. Interpret V8 je šířen jako opensource pod BSD licencí.

Firefox V prohlížeči nadace Mozilla Foundation, který se oficiálně nazýva Mozilla Firefox. V této nadaci bylo spousty enginů. Autor Brendan Eich a tvůrce JavaScriptového jazyka vymyslel interpret SpiderMonkey, interpret je kompletně napsán v jazyce C a byl využíván ve starších verzích tohoto prohlížeče. Je označován jako jeden z nejpokrokovějších enginů pro zpracování. Od verze Firefox 3.5 je používán jako engine TraceMonkey je proti svému předchůdci dvakrát rychlejší [7].

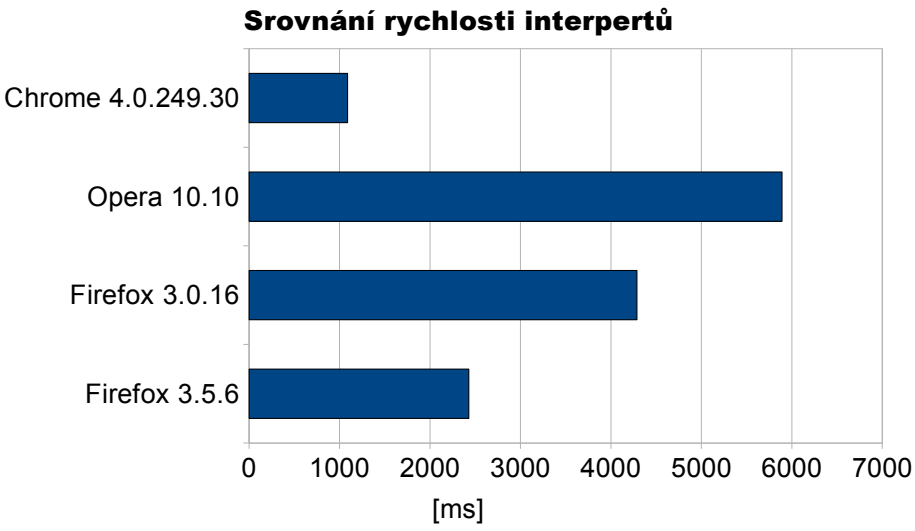
Opera Společnost Opera Software, která stojí za vývojem nyní již bezplatného a volně šířitelného webového prohlížeče Opera, ve verzi 10.0 využívá engine Futhark. Od verze 10.5 je plánován přechod na nový a výkonější engine s názvem Carakan, který bude využívat například generování do nativního kódu, automatickou objektovou klasifikaci a další technologie, které umožní dosáhnou vyšší rychlosti zpracování.

Jak je patrné z výčtu nejpoužívanějších prohlížečů, tak mezi interprety jednotlivých prohlížečů jsou rozdíly. Každý prohlížeč má svůj interpret. Tyto interprety se liší zejména svou rychlostí z hlediska provádění kódu. V čem by se ovšem neměly lišit je dodržování

standardů definovaných ECMA. Bohužel v určitých případech zde nastávají deinterpretace, které mohou vést k nepřesnostem či nefunkčnostem, což značně ztěžuje vývoj aplikací. Při programování pro známou množinu prohlížečů je možné tyto rozdíly programově obcházet. Avšak skutečnost je taková, že obcházení a zjišťování jak se daný kód interpretuje na klientské stanici v různých prohlížečích je časově zdlouhavé a náročné.

3.2.2 Test rychlosti interpretů

Pro srovnání interpretů jsem provedl ve všech prohlížečích nezávislý test rychlosti zpracování. Využil jsem již hotového benchmarku [19]. Výsledek testu je znázorněn v grafické podobě na obr.3.2.



Obrázek 3.2: Výsledky testu rychlosti pro jednotlivé webové prohlížeče.

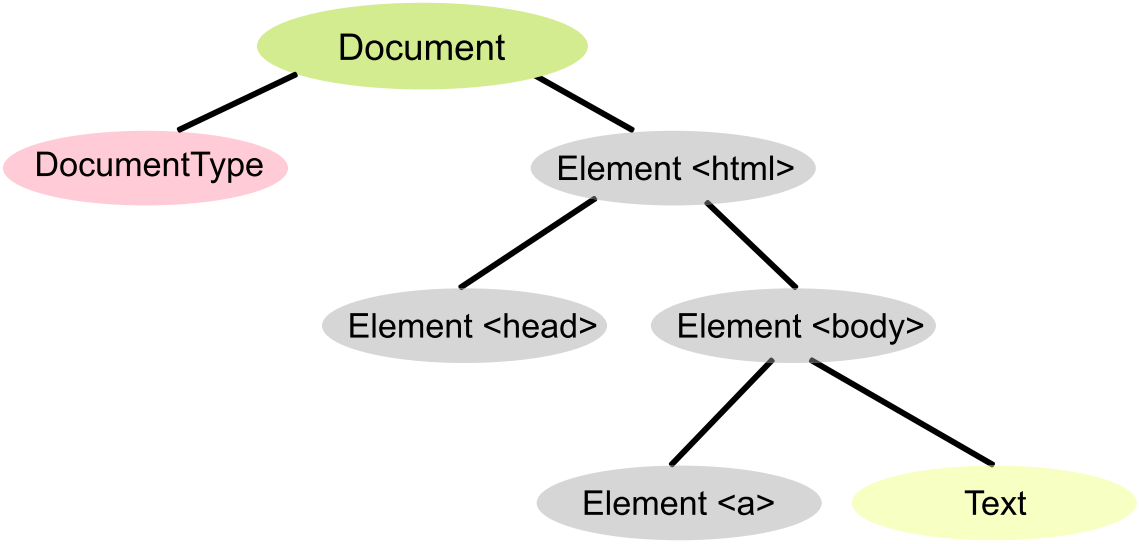
3.3 DOM

DOM (Document Object Model) je objektový model dokumentu HTML či XML. Jedná se o jazykově neutrální rozhraní, které umožňuje dynamický přístup programům a skriptům, využívaný například k modifikaci obsahu, struktur a stylů dokumentů [3]. Využívá koncepce objektově orientovaného programování a je platformě nezávislý. Umožňuje do dokumentu přidávat, manipulovat a vyhledávat prvky (elementy). Tento přístup je dnes aplikovaný v celé řadě JavaScriptových knihoven a je využíván k modifikacím v dokumentech HTML.

Na DOM lze nahlížet také jako na stromovou strukturu, jejímž kořenovým uzlem je uzel typu Document, který reprezentuje dokument jako celek. Příklad takové stromové struktury je znázorněn na obr. 3.3.

3.3.1 Atributy a Metody DOM

Specifikace DOM obsahuje i řadu atributů a metod, které lze využít při práci pomocí rozhraní. Výčet atributů je proveden v tabulce: 3.1.



Obrázek 3.3: Příklad stromového znázornění DOM.

`atributes`	atributy tohoto uzlu
`childNodes`	pole dceřiných uzlů
`documentElement`	kořenový uzel dokumentu
`firstChild`	první dceřiný uzel
`lastChild`	poslední dceřiný uzel
`localName`	lokální název uzlu
`name`	název uzlu včetně jmenného prostoru
`nextSibling`	následující příbuzný uzel
`nodeName`	název uzlu
`nodeType`	typ uzlu
`nodeValue`	hodnota uzlu
`previousSibling`	předchozí příbuzný uzel

Tabulka 3.1: Výčet atributů DOM [27]

Kromě atributů jsou definovány také metody, které umožňují operace s uzly. Mezi metody patří například `replaceNode(a, b)`, která nahradí uzel `b` uzlem `a`, `insertBefore(a, b)` která vloží uzel `a` před uzel `b` a `appendChild(a)` která připojí uzel `a` k uzlu, jehož metoda `appendChild` vyvolala.

3.3.2 Událostní model

Událostní model umožňuje reagovat na uživatelské vstupy. Objekt `event` definuje události na které můžeme programově reagovat. Existují desítky událostí, na které je možné vytvářet reakce. Ty, které zmíním byly vytvořeny a navrhnuty konsorciem W3C. Avšak v Microsoft Internet Exploreru, jako jedinném prohlížeči byla definována rozšiřující sada událostí. Jedná se například o události `copy paste`, které mají možnost reagovat na změnu obsahu uživatelovi schránky. Dále se jedná o události `drag`, `drop`, kde se sledují držení a puštění levého tlačítka myši. Podpora těchto rozšířených standardů se v ostatních prohlížečích může lišit a jejich funkčnost nemusí být zaručena. Pro přehled jsem rozdělil pouze události W3C do čtyř hlavních skupin.

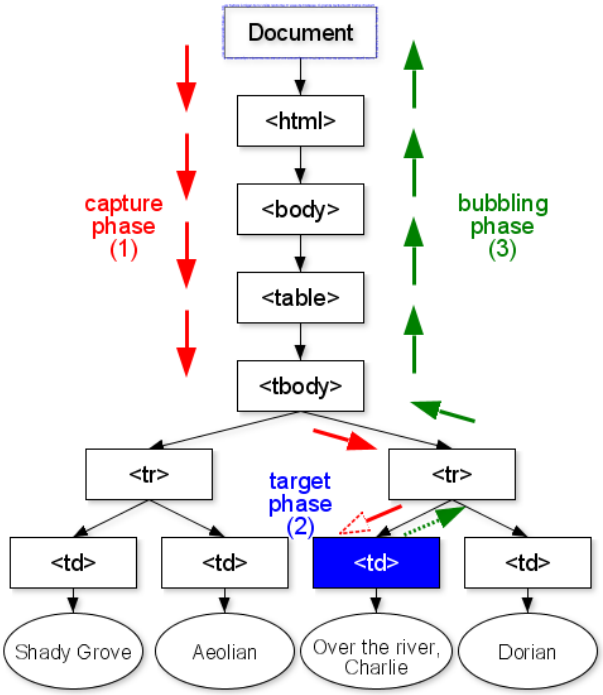
- události okna a dokumentu - např. `load`, `unload`
- události myši - např. `click`, `dblclick`, `mouseover`, `mousedown` , `mouseup`
- události klávesnice - např. `keypress`, `keydown`, `keyup`
- události formuláře - např. `submit`, `change`, `reset`

Události okna a dokumentu se věnují stránce samotné, sledují stav načtení stránky (`load`) a stav kdy uživatel opouští stránku (`unload`). Události myši je možné rozdělit do dvou kategorií: události, které sledují na jaké pozici se myš aktuálně nachází (`mouseover`) a události, které sledují kde bylo myší provedeno kliknutí (`click`, `mousedown`, `mouseup`) případně dvojkliknutí (`dblclick`). Události klávesnice sledující stisknutí kláves v rámci celého kontextu webové stránky. Klávesové události umožňují reagovat např. na klávesové zkratky, případně je lze využít při detekci vstupů do formulářů apod. Poslední kategorií jsou události formuláře. Zde se sleduje je možné reagovat po stisknutí tlačítka submit a provést například kontrolu formuláře, zda je vyplněn správně nebo reagovat na změny provedené v daném formuláři.

3.3.3 Registrace a zpracování událostí

Dle konceptu W3C konsorcia je vyžadováno událost nejprve zaregistrovat. Registrace se provede příslušnou funkcí `addEventListener`, která má tři parametry. První parametr udává druh události, druhý parametr určuje funkci, která se vyvolá jako reakce na zadanou událost. Posledním parametrem je hodnota typ bool, která určuje zdali reakce na vybranou událost bude provedena ve fázi zanořovací (capture phase) nebo vynořovací (bubbling phase). K odebrání registrace na událost je využito funkce `removeEventListener` opět se stejnými parametry.

V případě, že na jeden prvek je registrováno více událostí a není rozčleněno ve které fázi se bude reakce na událost provádět. Nelze jednoznačně rozlišit, zda se reakce na událost1 zpracuje dříve než reakce na událost2 a naopak. Navíc registrace událostí může probíhat v různých prohlížečích nejednotně a mimo standardy definované W3C. Existuje totiž koncept firmy Microsoft, která vytvořila svůj návrh na registraci a zpracování událostí. Jedním z hlavních rozdílů je, že reakce na událost probíhá vždy ve fázi „bubbling phase“.



Obrázek 3.4: Zpracovaní událostí, převzato z [4]

Na obrázku 3.4 je znázorněn tok události definovaný standardy W3C. Tento tok se skládá ze tří částí:

1) „capture phase“ - V této fázi se událost šíří od kořenovéhu uzlu `Document` a postupně se zanořuje do objekového modelu dokumentu. Postupně prochází jednotlivé prvky a pokud se mezi kořenovým uzlem a uzlem, který vyvolal příslušnou událost nachází jiný uzel, který má registrovanou reakci na stejnou událost a fázi zpracování jako je událost uzlu, který šíření události vyvolal, je tato nalezená událost také vyvolána.

2) „target phase“ - Stav, kdy je událost zaslána danému uzlu.

3) „bubbling phase“ - Fáze, která postupuje směrem od uzlu, který událost vyvolal až ke kořenovému uzlu `Document` objektového modelu dokumentu. Jedná se tedy o šíření události směrem vzhůru ke kořenovému uzlu. Opět i zde je možnost, že při šíření se může vyskytnout jiná registrovaná událost, se stejnými parametry jako jsou parametry uzlu, který událost vyvolal, a ta může být vyvolána.

3.4 Ajax

Ajax (Asynchronous JavaScript and XML) je jedna z technik využívaná při tvorbě dynamického webu. Je založen na JavaScriptu a XML. Principem Ajaxu je vytvoření objektu `XMLHttpRequest` jehož prostřednictvím je možné přijímat a zasílat asynchronní dotazy na server.

Postup při komunikaci:

1) Uživatel provede požadavek na server prostřednictvím HTTP protokolu a metody GET případně POST (dle velikosti a nároků požadavku).

2) Server zpracuje požadavek a po zpracování vrací odpověď.

3) Klient obdrží odpověď a ta může být dále vyhodnocována.

Interaktivní webové aplikace totiž nepotřebují kompletní aktualizaci celé stránky. Pokud by se měla provést například aktualizace konkrétního řádku v tabulce, není potřeba získat další informace potřebné pouze při inicializaci celého dokumentu.

Technika AJAX zahrnuje technologie:

- HTML a CSS pro prezentaci obsahu,
- DOM k modifikaci struktury dokumentu,
- XMLHttpRequest pro provedení asynchronního požadavku na server,
- XML a XSLT pro výměnu dat,
- Javascript, který výše uvedené technologie spojuje.

3.5 JavaScriptové knihovny

Jak jsem již zmínil v kapitole o CSS 3.1 a JavaScriptu 3.2, tak vývoj klientských aplikací je obecně zdlouhavý a časově náročný. Vývojáře aplikací mohou zpomalit především rozdíly v interpretaci výsledků jednotlivých webových prohlížečů a to zejména při vývoji větších projektů. Proto také vývoj aplikací na klientské straně směřuje k využívání knihoven jazyka JavaScript. Knihovny zahrnují kompletní potřeby programátora při vývoji. Zahrnují oblast značkovacího jazyka HTML, grafické reprezentace dokumentu (CSS), jazyka JavaScript a AJAX pro využití asynchronní komunikace se serverem. Vytváří nad využitými programovacími jazyky a technikami novou vrstvu. Ta má za úkol skrytí implementačních detailů, zefektivnění vývoje a programové ošetření z hlediska použití v konkrétním webovém prohlížeči. Knihovny obecně zahrnují podporu:

- AJAX - pro možnost asynchronní komunikace mezi klientem a serverem,
- DOM - pro přístup k objektovému modelu dokumentu a pohodlnou manipulaci s prvky,
- Událostí - pro jednotnou registraci a správu.

Rozsáhlejší knihovní celky také umožňují podporu:

- Pokročilých prvků GUI - dialogů, kontextových menu, záložek, formulářových prvků,
- Drag & Drop
- Animací a grafických efektů - jednoduché i pokročilé animace,
- Jazykovou podporu - která může zahrnovat desítky národních a jazykových specifikací.

3.5.1 Výhody a nevýhody

Uvážím-li možnosti využití knihoven, je nutné se nejprve zamyslet, jestli jejich použití povede k jednoznačným výhodám, či nikoliv. Využití pouze malé oblasti funkcí, ze značně rozsáhlých knihovných celků bude mít pravděpodobně pouze nevýhody, jelikož bude docházet k načítání velkých knihovních celků do paměti prohlížeče. Aplikace se tak zpomalí. Proto se knihovny nejčastěji využívají v rozsáhlých projektech, kdy se samotní programátoři nechtějí již zabývat implementačními detaily, ale raději použijí již předvytvořenou knihovní sadu. Knihovny z velké části zaručují jednotnou reprezentaci výsledné aplikace ve většině dnes využívaných internetových prohlížečů, to je dle mého názoru jedna z největších výhod. Výhodou je také správa událostí, kde je možné snadno událost zaregistrovat a knihovna samotná definuje pořadí jednotlivých akcí při provádění reakce na konkrétní vyvolanou událost. Další klíčovou výhodou je modularita. Ta umožňuje samotným programátorům přebírat již hotové funkčí celky nebo je samostatně vytvářet. Jednotlivé moduly je poté možné spojovat a definovat vzájemnou interakci. Větší knihovní celky samozřejmě obsahují i jazykové a národní specifikace. V těch jsou ustanoveny pravidla pro reprezentaci v příslušné světové oblasti jako je např. definice měn, hodinových formátů a zvyklostí jako je např. začátek nového kalendářního týdne nedělí nebo psaní zprava doleva v arabských státech.

Mezi nevýhody bych jednoznačně zmínil rychlost provádění aplikace při použití nativních volání. Jelikož použití knihovny vytvoří další logickou vrstvu a odstiňují tak nižší implementační vrstvy, bude rychlost provádění nižší. Vývojáři knihoven se problém rychlosti snaží minimalizovat a zdrojový kód co nejlépe optimalizovat. Mezi další problémy zajité patří rychlost načteční připojených knihovních celků, ať už se jedná o modul nebo jádro knihovny. Jedná se totiž o rozsáhlejší soubor metod a funkcí. Knihovní celek tak může zabírat desítky až stovky kilobajtů dat zdrojového kódu. Načtení a hlavně zpracování takového celku trvá cenné milisekundy. Aby nebylo nutné přenášet velké množství dat, je prováděna tzv. obfuskace kódu. Kdy jsou odstraněny přebytečné komentáře a bílé znaky jako mezery, tabulátory a další syntakticky nepotřebná data. Obfuskace umožní dosáhnout vyšší rychlosti načtení i na pomalém komunikačním spojení. Rychlost zpracování zdrojového kódu webovými prohlížeči se již také zvyšuje. Jak bylo řečeno v kapitole 3.2.1 o JavaScriptu a jednotlivých prohlížečích. Webové prohlížeče jsou schopny uchovat načtené knihovní celky ve vyrovnávacích pamětech. Není tedy pokaždé nutné provádět zpracovaní knihovního celku.

Kapitola 4

Představení zvolených knihoven pro implementaci

4.1 Dojo

První knihovnou, kterou využiji pro implementaci je knihovna Dojo. Jedná se objektově orientovanou knihovnu jazyka JavaScript.

Knihovna Dojo se skládá ze tří projektů:

- dojo,
- dijit,
- dojox.

První část, a to projekt **dojo**, obsahuje jádro knihovny. To definuje metody pro práci s objektovým modelem dokumentu (DOM), registrace a správu událostí, asynchronní komunikaci AJAX a podporu modulárního systému. Modulární systém umožňuje do stávajícího projektu umístit referenci na modul. Při spuštění kódu se do výsledného projektu postupně načtou všechny referencované moduly. Příslušný modul umožňuje načíst metoda `dojo.require`, která má jeden parametr. Ten udává relativní cestu, vzhledem k umístění jádra `dojo.js`. Relativní cesta je řetězec kde místo lomítka je použitým oddělovačem tečka a koncovka daného modulu se také neuvádí. Načtení modulu `dojo/fx.js` je tedy možné provést tímto zápisem: `dojo.require('dojo.fx')`. Další důležitou metodou modulárního systému je metoda `dojo.provide`. Prostřednictvím této metody je možné spouštěcímu systému jádra sdělit název nového modulu. Metoda akceptuje pouze jeden parametr a to řetězec s jednoznačným identifikátorem modulu. Pro použití nově definovaného modulu v modulárním systému je pak nutné prostřednictvím metody `dojo.registerModulePath(jmenný prostor, cesta)` definovat jmenný prostor.

4.1.1 Deklarace tříd

Jádro také obsahuje metody pro definici nových tříd. Na příkladu 4.1 je taková definice demonstrována. Metoda `dojo.declare()` má dva parametry, prvním je název třídy, druhým parametrem je výčet tříd, ze kterých se bude dědit. Výčet může být `null` nebo obsahovat až `n` odkazů na předky. Knihovna dojo tedy podporuje vícenásobnou dědičnost. Na příkladu

jsem demonstroval vytvoření konstruktoru (identifikátor `constructor`). Dále pak třída samozřejmě může obsahovat proměnné a vlastní metody. Vytvoření nové instance se provede za pomocí klíčového slova `new`, tedy v tomto případě `new moje.trida(param1);`.

```
dojo.declare("moje.trida", [předek1, předek2..], {
  cislo : 5, // vlastnosti třídy
  retezec : "string",

  constructor: function(param1) { // konstruktor třídy
    this.novyObjekt = new mujObjekt();
  },

  metoda: function(param1, param2) {
    ...
    ..
  }
});
```

Listing 4.1: Příklad deklarace třídy v Dojo

4.1.2 AJAX

V jádru se nachází také podpora XHR neboli AJAX. Výčet metod pro možnost komunikace jsem provedl v tabulce 4.1. Metody se dělí hlavně dle způsobu zasílání požadavku na server. Knihovna samozřejmě podporuje způsob zasílání typu GET, POST a PUT. Každá metoda přitom umožňuje nastavení parametrů, jejich výčet jsem provedl v tabulce 4.2.

Jméno metody	**Popis**
dojo.xhr(metoda, args, hasBody)	zašle požadavek dle nastavení metody (String) na HTTP server, parametr hasBody (boolean) udává, zda bude obsah připojen do těla zprávy
dojo.xhrGet(args)	zašle požadavek za pomocí HTTP GET metody
dojo.rawXhrGet(args)	zašle požadavek za pomocí HTTP GET metody společně s daty připojenými do těla zprávy
dojo.xhrPost(args)	zašle požadavek za pomocí HTTP POST metody
dojo.rawXhrPost(args)	zašle požadavek za pomocí HTTP POST metody společně s daty připojenými do těla zprávy
dojo.xhrPut(args)	zašle požadavek za pomocí HTTP PUT metody
dojo.xhrDelete(args)	zašle požadavek za pomocí HTTP DELETE metody

Tabulka 4.1: Výčet metod knihovny pro realizaci AJAX [26, s.118]

4.1.3 Manipulace s DOM

Pro práci s DOM jsou v jádře definovány další metody. Jde o metody pro vytvoření a zrušení prvku `dojo.create(tag, attrs, refNode, pos)` a `dojo.destroy()`. Metoda create má tři povinné parametry. Prvním je název vytvářeného prvku nebo reference na již existující DOM uzel, dále pak hashový objekt obsahující atributy prvku a posledním povinným

Jméno	Typ	Popis
url	String	URL adresa na kterou je požadavek směrován
timeout	Integer	počet milisekund, po kterých se čeká na odpověď
form	DOMNode/ String	počet milisekund, po kterých se čeká na odpověď
handleAs	String	specifikace formátu obržených dat (text, json, xml)
load	Funkce	funkce, která se vykoná po obdržení dat
error	Funkce	funkce, která se vykoná při chybě přenosu

Tabulka 4.2: Výčet parametrů asynchronního požadavku [26, s.117]

parametrem je uzel, který se poté předá metodě `dojo.place()` k umístění nově vytvořeného uzlu do existující struktury dokumentu. Posledním volitelným parametrem je parametr pos. Ten určuje, jaké umístění má metoda `dojo.place()` zvolit při připojení prvku do DOM. Možnosti jsou: „first“,„after“,„before“,„last“, „replace“ nebo „only“. U metody `dojo.destroy()` se předává pouze jeden parametr a tím je řetězec nebo objekt specifikující příslušný uzel DOM.

Další metody jádra se nacházejí v jmenném prostoru `dojo.*` a ten je specifikován v samostatném souboru `dojo.js`. Mezi podstatné metody zmíním medodu `dojo.addOnLoad`, která sleduje stav načtení objetového modelu dokumentu a ihned poté, co jeho generování je dokončeno spouští provádění. Kdyby příslušná metoda neexistovala, docházelo by k neočekávaným chybám. Narozdíl od události `onload` umožní spuštění provádění ihned poté, co je objektový model připraven, a to bez nutnosti stažení referencí na externí soubory připojených v dokumentu.

4.1.4 Dijit

Druhou částí knihovny je **dijit**, jedná se o knihovnu komponent GUI, které s v terminologii Dojo označují jako Widgety [26, s.5]. Ve jmenném prostoru `dijit.*` se nachází sada komponent předpřipravených k užití. Pro běh komponent je nezbytný přístup k jádru a jeho základním funkcím. Knihovní část je možné ještě dále rozdělit na:

- formulářové komponenty (`dijit.form.*`),
- layouty určující vzhled aplikace (`dijit.layout.*`),
- aplikační komponenty (`dijit.*`).

Formulářové komponenty obsahují rozšířené vlastnosti pro formulářové prvky z HTML. Nachází se ve jmenném prostoru `dijit.form`, pro zkrácení dlouhých názvů budu již tuto předponu v dalších částech vynechávat. Všechny prvky jsou stylované zvoleným tématem knihovny a mají tedy jednotný vzhled. V současné verzi dojo se nachází celkem čtyři barevná témata pod názvy: Claro, Tundra, Nihilo a Soria. Pro správnou reprezentaci je nutné v projektu vybrat jedno téma, které bude použito jako téma celé aplikace. Základem každého formuláře jsou vstupní boxy. Dojo poskytuje několik modifikací vstupních boxů dle jejich využití. Nachází se zde vstupní boxy pro výběr data (`DateTextBox`) při kterém se zobrazí pro výběr jiná komponenta a to konkrétně kalendář `dijit.Calendar`, času (`TimeTextBox`) kdy se zobrazí seznam časových hodnot pro výběr času, dalším vstupním prvkem je `CurrencyTextBox`, kterému se nastaví požadovaná měna, která bude zobrazena.

Dalšímy rozšířenými prvky jsou také tlačítka, která umožňují přidávání ikon, zaškrtávací pole a pole pro výběr jednoho prvku z n.

Dojo také rozšiřuje pojem layoutů. V jmenném prostoru `dijit.layout` lze v současné verzi naleznout celkem šest verzí layoutů. Utvoření rozmístění aplikace za pomocí layoutů je velmi vhodné. Layouty zajistí rozložení ostatních komponent do tzv. kontejnerů. Mezi layouty patří např. záložkový kontejner (TabContainer), oddělovací kontejner (BorderContainer), zásobníkový kontejner (StackContainer) aj. Dojo také umožňuje vytvoření vlastního layoutu s využitím třídy `dijit.layout._LayoutWidget` ta poskytuje základní vlastnosti jako:

- přidání komponenty - addChild(),
- odebrání komponenty - removeChild(),
- a další metody potřebné k vytvoření layoutu.

Poslední částí dijit je část obsahující aplikační komponenty. V této části se náchází řada spolu nesouvisejících komponent, která je vhodná pro využití ve specifických projektech. Pro představu o oblasti jsem provedl výčet komponent:

- dijit.Calendar - lokalizovaný měsiční kalendář, který umožňuje procházení po měsících a letech k vybrání data,
- dijit.ColorPalette - barevná paleta, sloužící k vybrání barvy,
- dijit.Dialog - dialogové okno sloužící k nastavení nebo zobrazení chyb,
- dijit.Editor - WYSIWYG editor,
- dijit.InlineEditBox - widget, který se po kliknutí změní na editovatelné pole,
- dijit.Menu - menu, zobrazené po kliknutí pravého tlačítka myši,
- dijit.MenuBar - typické menu známe z desktopů,
- dijit.ProgressBar - zobrazuje odezvu z aplikace, např. načítání apod.,
- dijit.TitlePane - panel, který se po kliknutí rozbalí,
- dijit.Toolbar - navigační lišta,
- dijit.Tooltip - tooltipy, které zobrazí např. nápovědu při najetí na konkrétní položku,
- dijit.TooltipDialog - tooltip s možností zadání údajů,
- dijit.Tree - hierarchická stromová struktura.

4.1.5 Dojox

Poslední částí je **dojox**. Obsahuje další různá rozšíření která se vyvíjí a jsou dostupná například i v experimentální podobě. V současné verzi 1.4 se zde nachází komponenty pro podporu webových galerií jako Google Picasa, nahrávání souborů na server, grafických knihoven a sledování přístupů za pomocí Google Analytics.

4.1.6 Datové sklady

Pro zjednodušení datové struktury pro naplňování složitějších komponent, jako hierarchických stromů a tabulek je v knihovně dojo vytvořen koncept tzv. datových skladů. Pro naplnění komponenty daty je nutné nejprve zvolit vhodný datový sklad. Jejich přehled jsem provedl v tabulce 4.3. Výběr datového skladu se provádí podle vstupních dat, která do něj chceme umístit a dle přístupnosti datového skladu. Po vytvoření datového skladu je nutné předat komponentě jeho referenci. Některé komponenty vyžadují ještě datový model, který bude reprezentovat data. V tomto případě se předává komponentě reference na datový model.

název datového skladu	**podporuje**	**formát dat**
dojo.data.ItemFileReadStore	čtení	JSON
dojo.data.ItemFileWriteStore	čtení/zápis	JSON
dojox.data.CsvStore	čtení	CSV oddělený znakem
dojox.data.OpmlStore	čtení	OPML
dojox.data.HtmlTableStore	čtení	HTML formátované tabulky
dojox.data.XmlStore	čtení/zápis	XML
dojox.data.QueryReadStore	čtení	JSON (AJAX dotazy na server)
dojox.data.AtomReadStore	čtení	Atom XML

Tabulka 4.3: Porovnání datových skladů Dojo dle přístupnosti a formátu zdrojových dat.

4.2 jQuery UI

Knihovna jQuery UI, kterou jsem vybral je tvořena v základu knihovnou jQuery rozšířenou o speciální funkce jako Drag & Drop, animací a různých doplňujících komponent. Důležitou vlastností jQuery je modularita. Pro knihovnu existuje velké množství komponent třetích stran. Knihovna samotná není totiž příliš obsáhlá a proto je nutné využívat projekty třetích stran, které rozšiřují základní jQuery objekt a přidávájí mu nové metody a komponenty. Na webu projektu se nachází databáze pluginů, které je možné s jQuery využívat. V současné době čítá rovných 4773 projektů. Knihovna se skládá z API rozhraní, které je rozděleno na:

- jádro jQuery,
- Manipulace s DOM,
- CSS,
- Události,
- Efekty,
- AJAX,
- jQuery UI.

V jádru knihovny se nachází pouze nezbytné funkce pro chod celé knihovny. Zmíním funkci `jQuery( výraz, [kontext] )`, která za pomocí využití CSS selektorů dokáže z objektového modelu vybrat množinu prvků, na kterou se budou další metody odkazovat a budou je případně modifikovat. První parametr funkce je tedy výraz, který určujeme selektorem CSS, výčet důležitých selektorů CSS jsem provedl v tabulce 4.4. Druhým parametrem

funkce je volitelný kontext, to může být například DOM prvek, dokument nebo jQuery objekt. Jádro také obsahuje sadu metod, které slouží k zjišťování vlastností a k přístupu do objektu jQuery.

selektor	syntaxe	funkce
univerzální selektor	`*`	vztahuje se na všechy prvky, většinou se vynechává
typový selektor	`X`	všechny prvky X
selektor třídy	`X.trida`	všechny prvky X s atributem `class=,,trida‘‘`
ID selektor	`X#xid`	prvek X s jedinečným identifikátorem `id=,,xid‘‘`
selektor následníka	`X Y`	všechny prvky Y, které jsou uvnitř prvku X
selektor dítěte	`X>Y`	všechny prvky Y, které jsou dítětem X
selektor sourozenců	`X+Y`	všechny prvky Y, které mají stejného rodiče jako X
atributové selektory	`X[attr]`	všechny prvky X, které mají atibut attr

Tabulka 4.4: Základní výčet selektorů CSS2

Manipulace s dokumenotvým modelový objektem je velmi snadná. Existují zde metody pro filtrování. Mezi ně patří např. `jQuery.eq()`, která umožní vybrat n-tý prvek z konkrétní množiny o n prvcích. Dále pak metoda `jQuery.map()`, která umožňuje vyfiltrování atributů a hodnot atributů a převádět je na pole jQuery objektů.

Knihovna podporuje také AJAX. Pro provedení GET nebo POST asynchronního požadavku na server jsou definovány dvě metody `jQuery.get` a `jQuery.post`. Obě metody mají shodné tři parametry, kde první z nich udává url adresu, na kterou bude požadavek zaslán. Druhým parametrem je v případě metody `get` řetězec v JSON notaci, který udává parametry ve tvaru klíč:hodnota. V případě medoty `post` se udávají data ve formátu řetězce, která budou prostřednictvím dané metody zaslána na server. Poslední parametr je volitelný a v něm může být definována callback funkce, která se po odpovědi na požadavek vykoná. Kompletní výčet metod jádra knihovny pro práci s asynchronními požadavky jsem provedl v tabulce 4.5.

Jméno metody	**Popis**
jQuery.ajax()	provede asynchronní požadavek dle předaných parametrů
jQuery.get()	zašle požadavek za pomocí HTTP GET metody
jQuery.getJSON()	zašle požadavek za pomocí HTTP GET metody a JSON odpověď načte do pole
jQuery.getScript()	zašle požadavek za pomocí HTTP GET metody na script jazyka JavaScript a provede jej
jQuery.load()	načte data ze serveru a ta vloží do příslušného prvku html
jQuery.post()	zašle požadavek za pomocí HTTP POST metody

Tabulka 4.5: Výčet metod knihovny pro realizaci AJAX v jQuery

Načtení knihovny a případných modulů do projektu se provádí přímo v HTML dokumentu za pomocí tagu `script` vloženého do hlavičky dokumentu. Na příkladu 4.2 jsem provedl příklad připojení knihovny jQuery a jQuery UI do struktury HTML. Pod referencemi na zdrojové kódy jazyka JavaScript se nachází reference na CSS styly. Knihovna umožňuje připojení vždy jednoho stylu, který reprezentuje příslušné grafické téma. Grafické téma si uživatel vytváří sám nebo si vybere již z předvytvořených.

```
<!--Reference na základní knihovnu jQuery -->
<script type="text/javascript" src="./js/jquery-1.3.2.min.js"></script>
<!--Reference na rozšíření UI -->
<script type="text/javascript" src="./js/jquery-ui-1.7.2.custom.min.js">
</script>

<!-- Reference na css styl -->
<link type="text/css" href="./js/themes/ui-lightness/ui.all.css"
 rel="stylesheet">
<!-- Reference na css styl rozšíření UI -->
<link rel="stylesheet" type="text/css" media="screen"
href="./js/themes/ui-lightness/jquery-ui-1.7.2.custom.css">
```

Listing 4.2: Import knihovny jQuery

4.2.1 Rozšíření UI

Knihovna v rozšíření UI poskytuje oproti oběma srovnávaným knihovnám pouze základní funkce pro vývoj GUI. Lze je rodělit do třech kategorií:

- Interakce,
- Komponent,
- Efektů.

Rožšíření interakce poskytuje funkce jako Drag & Drop, změnu velikosti prvků, možnost vybírání a řazení prvků. Do komponent patří např. záložky, dialogy, tlačítka, posuvníky apod. Do efektů lze zařadit různé animace, které lze použít např. v kombinací s dialogovými okny, jako postupné zesvětlení/ztmavení. Efekty, které plynule mění barvy a vytváří „pulsující režim“ apod.

Rozšíření z hlediska komponent je pouze minimální. jQuery UI přidává komponenty typu tlačítek, záložek, dialogů, kalendářových komponent apod. Ve srovnání s knihovnami Dojo a Qooxdoo jde však pouze o jednoduché komponenenty, které nemusí vždy splňovat všechny kladené požadavky.

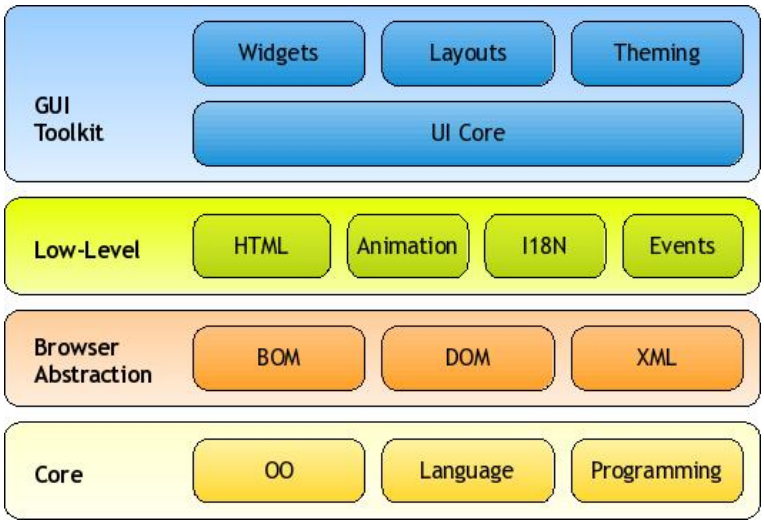
Zajímavým prvkem rozšíření UI je možnost využití motivů, které definují jednotný vzhled použitých komponent. Na oficiálních stránkách projektu lze k současné verzi naleznout celkem 24 již hotových motivů vzhledu. Na stránce [14] lze již hotové motivy naleznout a případně i dále upravovat. Stránka je interaktivní aplikací, kde je možné navrhnout osobní barevné schéma a na příkladech komponent je vytvářený motiv ihned zobrazen. Aplikace tedy v zásadě ulehčuje a zpřehledňuje tvorbu barevných schémat.

4.3 Qooxdoo

Další knihovnou, kterou jsem vybral pro implementaci je JavaScriptová knihovna Qooxdoo[ku:ksdu:]. Jedná se o otevřenou knihovnu pro pohodlný vývoj aplikací jako výše dvě zmíněné. Avšak Qooxdoo volí jiný přístup při tvorbě aplikace. Autoři knihovny si zakládají na tom, že vývoj programu by měl zvládnout programátor i bez zkušeností a znalostí HTML, CSS a dokonce ani DOM. Vývoj aplikace by se měl podobat spíše vývoji desktopové aplikace než aplikace pro web. Základem vývoje je tvorba komponent, případně je možné

využít specializované nástroje, které umožní vytvořit aplikaci v jiném jazyku (Java, .NET) a výsledek přeložit do jazyku JavaScript, který využívá této knihovny.

Knihovna umožňuje snadné využití objektově orientovaného programování v jazyce JavaScript. Knihovní jádro umožňuje vytvářet třídy, rozhraní, „mixiny“ a definuje jednoduchou dědičnost. Architektura Qooxdoo je vrstvově znázorněna na obr.4.1. Základem je jádro, které umožňuje vytvářet třídy, rozhraní, „mixiny“ a definuje jednoduchou dědičnost.



Obrázek 4.1: Znázornění architektury knihovny Qooxdoo, převzato z [10]

4.3.1 Deklarace tříd, mixinů

Deklaraci třídy je možné provést `qx.Class.define(name, config)` kde `name` je jméno nově vytvářené třídy a `config` umožňuje specifikovat, o jaký typ třídy se konkrétně jedná. Na výběr je klasická třída, která umožňuje definici proměnných, metod a instancí tříd. Dále pak statická třída, která je omezena pouze na proměnné a metody dané třídy, abstraktní třídy, které nepovolují vytváření instancí a unikátní třídy, které nedovolí spuštění více jak jedné instance objektu dané třídy.

Na příkladu 4.3 jsem uvedl příklad deklarace nové třídy. Identifikátor `extend` určuje předka, ze kterého je možné provést dědění. Z definice je patrné, že knihovna podporuje pouze jednoduchou dědičnost. Kontruktor třídy je specifikován pod identifikátorem `construct`. Statické proměnné, související pouze s danou třídou, se mohou vytvořit za použití identifikátoru `static`, tyto proměnné lze nejčastěji použít jako počítadla otevřených instancí apod. Vlastnosti a metody třídy se specifikují samostatně v `members`. Vytvoření instance se provede klasickým způsobem za použití klíčového slova `new`: `var instance = new moje.trida;`.

Mezi speciální koncepty knihovny patří „mixiny“. „Mixiny jsou specialitou objektově orientovaných programovacích jazyků. Obsahují proměnné a metody, které mohou být propojeny do jiných tříd. Jsou podobné třídám, ale nemohou vytvářet samostatné instance. Na rozdíl od rozhraní obsahují implementační kód. Typicky jsou tvořeny jen několika členy, které umožňují obecně provádění některých velmi specifických funkcí“ [17].

```
qx.Class.define("moje.trida",
{
  extend : předek,

  // statické proměnné
  statics : {
    pocetinstanci: 0,
  },

  // konstruktor
  construct: function()
  {
    this.vlastnost = "Hello world";
  },

  members :
  {
    vlastnost : "",

    getVlastnost: function()
    {
      return this.vlastnost;
    }
  }
});
```

Listing 4.3: Příklad deklarace třídy v Qooxdoo

4.3.2 Nástroje knihovny

Knihovna také obsahuje další nástroje, které vývojáři usnadní práci. Kompletní balík knihovny lze obdržet jako Software Development Kit (SDK). Jedná se o vývojové prostředí, které je možné využít k sestavování a pro snadnější vývoj aplikace. Balík SDK samozřejmě obsahuje kompletní zdrojové kódy knihoven. Dále zahrnuje sadu skriptů napsaných ve skriptovacím jazyce Python. Tyto skripty jsou nezbytné při vývoji a umožňují:

- vytvoření nového projektu,
- sestavení aplikace - spojení aplikace s třídami, které využívá,
- vytvoření testovacího prostředí - k definici testů pro danou aplikaci,
- vytvoření API dokumentace.

Vytvoření nového projektu je možné spuštěním skriptu `create-application.py` který se nachází v adresáři `tool/bin`. Následně je vytvořena kostra projektu, která obsahuje HTML webový dokument. V dokumentu je provedeno připojení základu knihovny. Dále je vytvořen skript `generate.py`, který umožní s projektem provádět další potřebné akce důležité při vývoji.

Po vytvoření nového projektu je nutné projekt spojit s třídami, které byly v projektu použity. Spojení aplikace se provádí příkazem `generate.py source-all` v kořenovém adresáři aktuálního projektu. Skript zajistí načtení všech tříd, ze kterých poté vybere ty,

```
qx.Mixin.define("jméno",
{
  include: [SuperMixins],

  properties: {
    "tabIndex": {check: "Number", init: -1}
  },

  members:
  {
    vlastnost: 0,
    metoda1: function() {},
    metoda2: function() {}
  }
});
```

Listing 4.4: Příklad definice mixinu v Qooxdoo

které byly v daném projektu využity. Je schopen vyřešit i určité vazby a závislosti mezi jednotlivými třídami. Dále pak vygeneruje zdrojový kód, který přísluší danému projektu. Po provedení je zaručeno, že daný projekt bude možné spustit. Velkou výhodou zpracování skriptem je fakt, že o spojení s třídami rozhoduje program a není tedy možné aby se v projektu nacházely nepoužité části knihovny, které by pouze zpomalovaly chod celé aplikace.

SDK obsahuje také sadu testů, které je možné využít při odlaďování chyb při práci s knihovnou. Po zadání příkazu `generate.py test` je do kořenového adresáře projektu vygenerována nová složka s názvem `test`. Ta obsahuje dokument `index.html`, ve kterém je vytvořeno rozhraní „Test runner“, to umožní aplikovat sadu testů na třídy v daném jmenném prostoru. Je možné si nadefinovat vlastní sadu testů, nad kterými bude výsledná aplikace testována.

Skriptem `generate.py` je možné vytvořit i lokální API dokumentaci. Do dokumentace se automaticky přidávají třídy vytvořené v rámci konkrétního projektu a dokonce i komentáře, které byly při psaní třídy použity. V dokumentaci nalezneme kompletní referenci všech jmenných prostorů knihovny a samozřejmě i jmenného prostoru daného projektu. Je vytvořen kompletní náhled na všechny třídy a rozhraní použité v daných jmenných prostorech. O všech komponentách lze zobrazit detailní informace od konstruktorů daných tříd, privátních proměnných, děděných metod až po metody, které jsou třídou definovány. Kompletní API dokumentaci daného projektu je tedy možné vytvořit příkazem `generate.py api`.

4.3.3 GUI Toolkit

Vrstva GUI Toolkit obsahuje nástroje pro tvorbu grafického uživatelského rozhraní. Na této vrstvě může uživatel pohodlně vytvářet prvky grafického rozhraní. Jednotlivé prvky jsou definované jako komponenty, jejichž vlastnosti je možné nastavovat. Při vývoji aplikace jsou úplně odstíněny nižší vrstvy jako HTML, CSS. Vývoj aplikace je velmi podobný vývoji desktopových aplikací. Dle mého názoru se podobá prostředkům pro tvorbu GUI v C++ jako je Qt (Qt je multiplatformní knihovnou, která umožňuje snadno vytvořit GUI pro program napsaný v jazyce C++).

`qx.ui.basic`	základní komponenty jako obrázky, štítky
`qx.ui.control`	kolekce „high-level GUI“ obsahující např. vybírání barev, kalendář.
`qx.ui.core`	jádro UI
`qx.ui.decoration`	dekorace použité ke stylizaci widgetů
`qx.ui.form`	jmený prostor s velkou množinou formulářů
`qx.ui.layout`	třídy pro definice layoutů
`qx.ui.menubar`	obsahuje třídy pro tvorbu desktopového menu
`qx.ui.table sada`	tříd pro práci s tabulkovými daty
`qx.ui.tree`	sada tříd pro zobrazení stromových struktur adresářů apod.
`qx.ui.window`	sada tříd pro vytváření oken

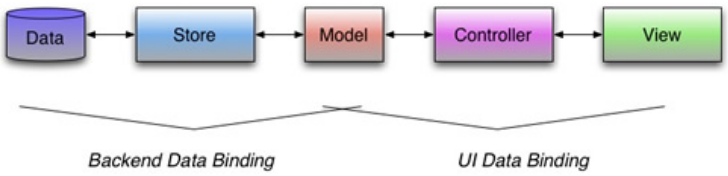
Tabulka 4.6: Výčet důležitých balíčků pro tvorbu GUI

GUI Toolkit vrstva nabízí sadu komponent, jejímž prostřednictvím je možné vytvářet např.: štítky (labels), tlačítka, textová pole, popup vyskakovací okna. Všechny tyto komponenty je možné vsazovat do layoutů. Ty se postarají o správné rozmístění v daném dokumentu. Postupným skládáním jednotlivých komponent, lze dosáhnout sofistikovaného grafického prostředí blízké prostředí desktopu. V GUI lze definovat také klávesové zkratky například pro jednotlivé položky menu. Na tyto zkratky je možné nastavit odchytávání a zpracování událostí.

Pro představu o jmenném prostoru `gui.ui` a možnostech GUI uvádím tabulku 4.6, která obsahuje výčet jmenných prostorů a přehled jaké druhy tříd lze v jednotlivých prostorech naleznout.

4.3.4 Datové sklady

Stejně jako knihovna Dojo, tak i knihovna Qooxdoo obsahuje koncept tzv. datových skladů. Obr. 4.2 znázorňuje hlavní myšlenku pro možnosti připojování dat prostřednictvím datových skladů. Jak je vidět na obrázku, tak připojování dat zahrnuje celkem pět komponent. Data - reprezentují soubor případně server, který poskytuje data. Store (sklad) - realizuje stažení a uchování dat pro možnost připojení. Datových skladů není takové množství jako je tomu v dojo, poskytuje pouze dva druhy: `qx.data.store.Json` pro uchování zdrojových dat ve formátu JSON a `qx.data.store.Jsonp` pro zdrojový formát dat JSON notaci, kdy je možné přebírat zdrojový kód i z jiného, než lokálního serveru. Prostředníkem mezi komponentami a datovými sklady je Model. Ten uchovává data a zprostředkovává interakci skladu a řadiče. Controller (řadič) spojí data z Modelu do zobrazovacích komponent. View pak reprezentuje koncovou komponentu, která je závislá na controlleru a přes něj přebírá a zobrazuje připojená data.



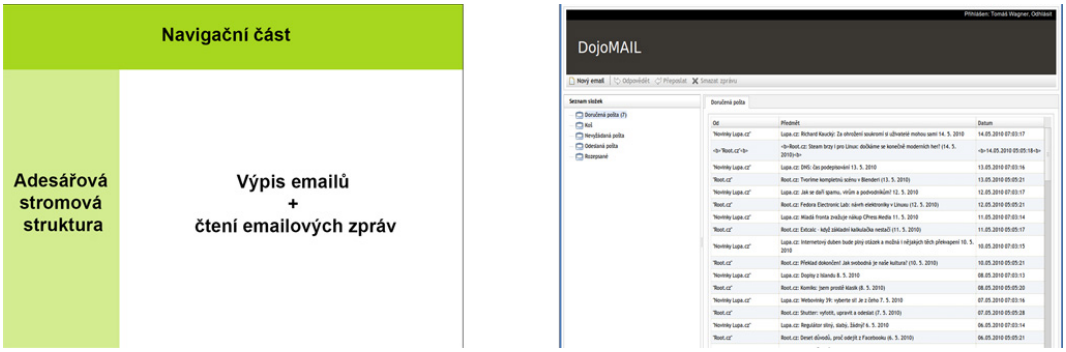
Obrázek 4.2: Znázornění datové vrstvy knihovny Qooxdoo, převzato z [8]

Kapitola 5

Návrh emailového klienta

5.1 Návrh GUI

Grafické uživatelské rozhraní je ve všech implementacích orientováno dle klasického schéma viz. obr. 5.1. Při návrhu jsem kladl důraz především na jednoduchost, snadnou použitelnost a orientovatelnost.



Obrázek 5.1: Návrh GUI

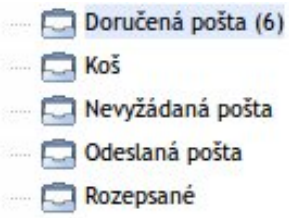
5.2 Přehled možností a nastavení emailového klienta

Navigační část poskytuje uživateli především informační oblast, kde se nachází název příslušného klienta a navigační panel poskytující základní funkcionalitu. Na panelu se nachází tlačítko pro vytvoření nového emailu. Po vybrání příslušné zprávy v seznamu zpráv jsou zde také možnosti pro operace s danou zprávou jako odpovězení, přeposlání a smazání. Na obr.5.2 je uveden názorný příklad panelu.

Obrázek 5.2: Navigační panel

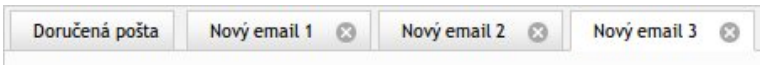
V **oblasti adresářové stromové struktury** definované na obrázku by mělo jít zejména o hierarchickou stromovou strukturu. Definuje složky a podsložky do určité úrovně zanoření. V případě hlubšího zanožení do stromové úrovně je zde možnost „zabalovat“ a „rozbalovat“ konkrétní podadresáře. Stromová struktura tedy reprezentuje adresářovou strukturu kdy

v jednotlivých adresářích se nachází příslušné emaily. Mezi základní adresáře patří: Doručená pošta, Odeslaná pošta, Rozepsané, Nevyžádaná pošta a Koš, tak jako na obr.5.3. U složky Doručená pošta se v závorce zobrazuje počet nepřečtených emailových zpráv.

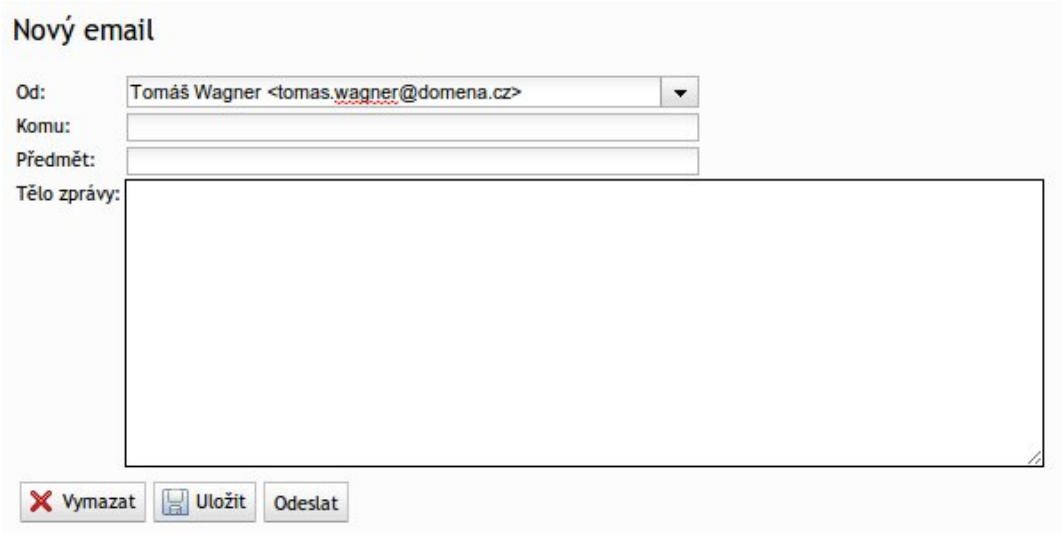


Obrázek 5.3: Základní adresářová struktura

Třetí oblastí je **oblast pro výpis emailových zpráv** příslušné složky. Oblast zahrnuje sadu záložek 5.4. První záložka je neměná a zobrazuje vždy výpis složky s emailovými zprávami. Dalších n záložek se vytváří kliknutím na tlačítko Nový email. Záložky s novým emailem obsahují formulář k odeslání nového emailu. Tento formulář jsem zobrazil na obr. 5.5. Základními poli jsou pole: odesílatel, adresát, předmět a obsah emailu. Pod samotným formulářem se nachází tlačítka pro práci s vytvořeným emailem. Nový email je možné úplně vymazat (vymazat formulář), případně uložit a samozřejmě odeslat.



Obrázek 5.4: Záložky okna



Obrázek 5.5: Záložka s formulářem pro nový email.

Kapitola 6

Implementace

Pro implementaci jsem využil knihoven v následujících verzích: **JQuery UI 1.7.2** rozšiřující standartní knihovnu **jQuery 1.3**, **Dojo Toolkit 1.4** a **Qooxdoo 1.0.1**. V době implementace nejnovějších verzí. Při implementaci bylo ze všech knihoven využito

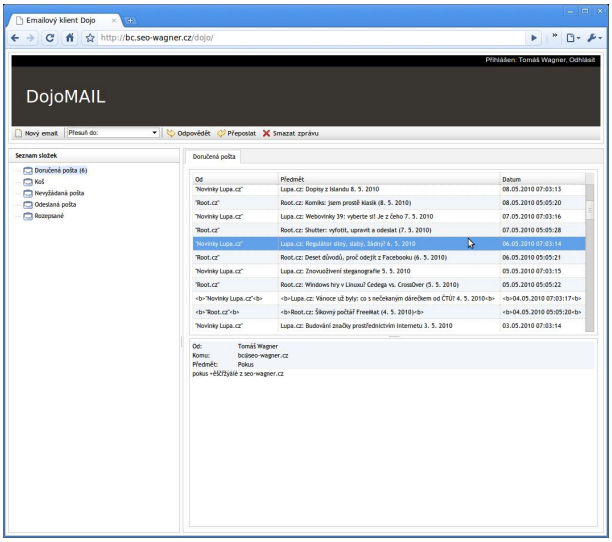
- barevných témat - pro vhodnou grafickou reprezentaci aplikace,
- layoutů - pro rozdělení aplikace do tří informačních částí,
- stromových komponent - pro vytvoření adresářové struktury znázorňující seznam složek,
- tabulkových komponent - pro výpis emailových zpráv,
- dialogů - pro zobrazení upozornění či chybových hlášení.

V dalších podkapitolách popíši způsob implementace za použití konkrétních knihoven. V popisu imlementace se nejprve u jednotlivých knihoven zaměřím na vlastnosti, které mi při implementaci nevyhovovaly a které konstrukce byly z hlediska implementace vhodné. Při tom se pokusím využít maximální možnosti dané knihovny, které poskytuje.

6.1 Dojo

Při implementaci emailového klienta s využitím knihovny Dojo jsem pro rozmístění jednotlivých oblastí využil layoutů. Layouty umožňují pohodlné rozdělení hlavního okna aplikace do konkrétních částí. Těmto částem lze pak nastavovat vlastnosti k chování např. při změně velikosti okna, při změně velikosti oblasti apod. Layouty také umožňují jednotlivým částem měnit velikost (šířku, výšku). Systém layoutů již přímo počítá s tím, že do jednotlivých částí přijde umístění konkrétních komponent. Dle základního návrhu je tedy okno webového prohlížeče, za použití layoutu `dijit.layout.BorderContainer`, rozděleno do třech částí.

Kostrou aplikace je HTML soubor `index.html`. V něm je za pomocí blokových prvků vytvořena kostra emailového klienta. Využil jsem také dodatečných dojo html atributů. Tyto atributy sice nesplňují specifikaci W3 konsorcia a nejsou obsaženy mezi povolenými v DTD hlavičce dokumentu ale jejich využití je velmi jednoduché a z rozmístění je pak na první pohled patrné. S pomocí atributu `dojoType` jsem přímo každému blokovém prvku nastavil konkrétní komponentu. V případě použití atributu `dojoType` je možné dalšími atributy měnit nastavení komponenty.



Obrázek 6.1: Vzhled aplikace v Dojo

6.1.1 Navigační oblast

Navigační část je tvořena ze dvou částí. První částí je informační část, ve které se nachází pouze název emailového klienta. Druhou částí je část, která obsahuje navigační panel. Ten je tvořen komponentou `dijit.Toolbar` ve které jsou umístěné tlačítka pro vytvoření nového emailu použitím komponenty pro tvorbu tlačítka `dijit.form.Button`. Každé tlačítko bylo doplněno o ikonu charakterizující využití. Pro účel vytváření nových tlačítek jsem definoval funkci `createButton (popisek,ikona)`, která přebírá dva parametry a to popisek tlačítka a název ikony, která bude na tlačítku viditelná. Funkce vrací nově vytvořený objekt. Dále se na panelu nachází tlačítka pro práci s emailovou zprávou. V případě, že nebyla vybrána žádná zpráva, jsou tato tlačítka deaktivována a neumožňují uživateli jejich stisknutí. Pokud již byla zpráva vybrána, je uživateli nabídnuta možnost na zprávu odpovědět, přeposlat nebo ji smazat. Navigační panel jsem zobrazil na obr.6.2.

Obrázek 6.2: Navigační panel dojo

6.1.2 Adresářová struktura

Pro adresářovou strukturu v projektu jsem využil třídy `dijit.Tree`, která umožňuje vytvořit stromovou strukturu pro zobrazení složek emailových zpráv. Koncept komponent jako stromových struktur, tabulek apod. je navržen tak, že všechny tyto komponenty přejímají tzv. datové sklady. Na příkladu 6.1 je vidět vzájemné propojení komponent. Komponenta `dojo.data.ItemFileReadStore` vytváří datový sklad, jeho reference je předána `dijit.tree.ForestStoreModel` a reference modelu je po-té předána `dijit.Tree`. Vzhledem ke vstupním datům ve formátu JSON byl zvolen datový sklad, který tento vstupní formát podporuje.

```
//definice úložiště stromu pro výpis emailů
var dirstore = new dojo.data.ItemFileReadStore({
  url: "../interface.php?action=getDojoDirs"
});

//definice modelu stromu pro výpis emailů
var dirmodel = new dijit.tree.ForestStoreModel({
  store: dirstore
});

//definice stromu pro výpis emailů
var dirtree = new dijit.Tree({
        openOnClick: true,
        model:dirmodel,
        showRoot:false,
  getIconClass: function(item){
    return "customFolderClosedIcon";
  }
});
```

Listing 6.1: Příklad definice stromové komponenty v Dojo

6.1.3 Výpis emailových zpráv

Výpis emailových zpráv je realizován za pomocí komponenty `dojox.grid.DataGrid`. Komponenta využívá stejně, jako v případě stromu adresářové struktury, datové sklady. Datovým skladem je opět `dojo.data.ItemFileReadStore`, jelikož vstupní data tabulky jsou z oblasti serveru předávána ve formátu JSON. Datový model u tabulky není nutné definovat, místo něj se definuje struktura, která specifikuje parametry každého sloupce tabulky. Parametry jsou např. id pole, jméno, šířka apod.

```
// vytvoříme tabulku pro výpis emailů
var emailgrid = new dojox.grid.DataGrid({
  store: emailstore,
  clientSort: true,
  selectionMode: 'single',
  rowToPage:10,
  structure: emaillayout,
});
```

Listing 6.2: Příklad definice tabulky pro výpis emailů v Dojo

6.1.4 Vytváření nového emailu

Pro vytváření nového emailu jsem v dojo vytovořil kostu nové zprávy, kterou jsem definoval jako komponentu s názvem `NewMessage(id)`, která dostává jedinný parametr a to id rozlišující jednotlivé nově vytvářené zprávy. Formulářové prvky byly také definovány prostřednictvím atributu dojoType aby odpovídaly prvkům Dojo GUI. Pro výběr identity odesílatele

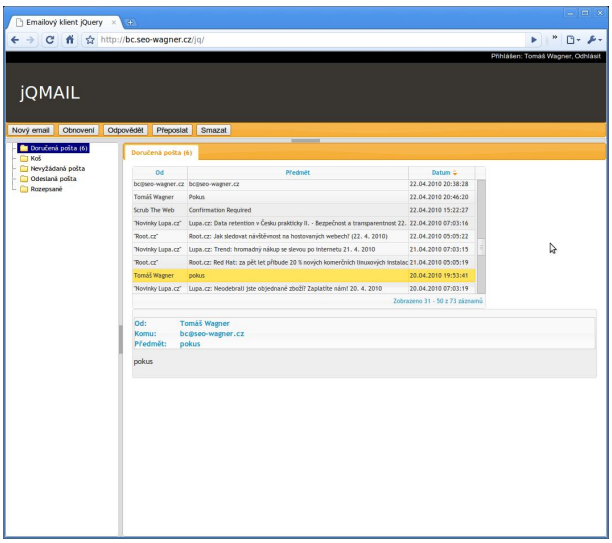
byl select předefinován na `dijit.form.ComboBox`, vstupní boxy na `dijit.form.TextBox`, textbox na `dijit.form.TextBox`. Tlačítka pro uložení byla také vytvořena jako `dijit.form.Button` a ke každému byla přidána ikona charakterizující význam.

Nový email se připojuje ve formě záložky do záložkového layoutu `dijit.layout.TabContainer`. Záložky s novými emaily jsou zavíratelné za pomocí křížku u jména konkrétní záložky.

6.2 jQuery

Abych zachoval stejné vlastnosti navrženého konceptu emailového klienta, bylo s použitím knihovny jQuery nutné využít další moduly. Knihovna s rozšířením UI, neobsahovala všechny komponenty, které jsem pro vývoj aplikace potřeboval. Byly nutné tři další komponenty z projektů třetích stran, které nespadají pod vývoj jQuery. Jedná se o komponentu stromu **jsTree 0.9.9a** [16], komponentu tabulky pro výpis emailů **jqGrid 3.6** [12] a komponenty pro vytvoření layoutu **jQuery UI Layout 1.2.0** [15]. Tato rozšíření jsou vyvíjená samostatně komunitami pohybujícími se okolo jQuery. Všechna rozšíření jsou distribuována pod volně šiřitelnými licencemi.

Pro rozmístění jsem využil layoutů stejně jako v knihovně Dojo. Layouty jsem stejně rozdělily oblast do třech částí tak jak je patrné z obr.6.3. Jednotlivým částem je možné upravovat velikost a skrývat je. Při zmenšování velikosti layoutu zde nedochází ke zmenšování vnořených komponent, jelikož jednotlivé komponenty mezi sebou nemají žádný vztah. Není tedy knihovnou definovano komunikační rozhraní komponent, které by ovlivňovalo jejich nastavení, tak jak tomu je například u knihovny Dojo.



Obrázek 6.3: Vzhled aplikace v jQuery

6.2.1 Navigační oblast

Navigační oblast je tvořena z hlavičky, která obsahuje označení klienta, a z navigačního panelu, který poskytuje možnosti pro vytvoření nového emailu a manipulacemi se stávajícími emailovými zprávami. Navigační panel je blokovým elementem, obsahující tlačítka z rozšíření UI. Prvním tlačítkem je tlačítko pro vytvoření nového emailu. Další tlačítka jsou stejná jako dle návrhu. A jedná se o tlačítka pro odpověď zprávy, přeposlání a smazání.

6.2.2 Adresářová struktura

Adresářová struktura byla vytvořena za pomocí jsTree rozšíření. Příklad 6.3 ukazuje definici pro vytvoření adresářové struktury. Strom se naplňuje za pomocí asynchronního požadavku na server a to na adresu `../interface.php?action=getJQDirs` jako formát dat byl vybrán JSON. Dále následuje definice callback funkcí, které souvisí s událostmi provedenými na konkrétní instanci stromu.

```
// vytvoření stromu adresářové struktury
$("#treemenu").tree({
  data : {
    type : "json",
    opts : {
      url : "../interface.php?action=getJQDirs"
    }
  },
  callback : {
    onchange : function (NODE, TREE_OBJ) {
            // vynecháno tělo
                },
                onload : function () {
            // vynecháno tělo
                }
        }
};
```

Listing 6.3: Vytvoření instance stromové adresářové struktury

6.2.3 Výpis emailových zpráv

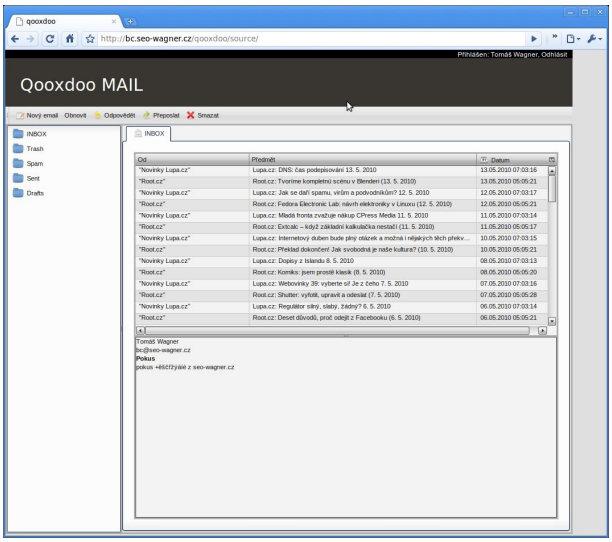
Aby bylo možné realizovat výpis emailových zpráv, bylo nutné vybrat vhodný modul knihovny jQuery. Knihovna bohužel komponentu tabulky v základu neobsahuje. Proto byl využit modul jqGrid, který umožňuje vytvoření komponenty tabulky. Nastavení vlastností tabulky je široké, proto ho zde z důvodů obsáhlosti neuvádím jako příklad. K naplnění daty dochází opět asynchronním požadavkem na server s příslušnými parametry určujícími stránku, počet záznamů apod. V implementaci byl z tabulky využit režim, kdy nedochází ke kompletnímu naplnění tabulky daty na jeden dotaz. Pokud je v tabulce více záznamů, reaguje na posuvník, který je umístěn napravo tabulky. Posunutím je vyvolána událost, při které se spočítá, na které stránce se nacházíme a vytvoří se opět dotaz na server pro zjištění tabulkových záznamů.

6.2.4 Vytváření nového emailu

Pro vytváření nového emailu byla v kostře projektu vytvořena html bloková struktura s formulářem, která je po načtení aplikace implicitně skryta. Vyvolání provádí až kód jazyka JavaScript a formulář vkládá do příslušné nově otevřené záložky.

6.3 Qooxdoo

Implementace v Qooxdoo byla z implementačního hlediska nejnáročnější. Je to dáno především přístupem a formou realizace projektu. Pro chod projektu je nutné společně se zdrojovými kódy aplikace mít sebou vývojový kit, na kterém je celý chod aplikace závislý. SDK Qooxdoo obsahuje 8856 souborů a jeho velikost činí 48MB.



Obrázek 6.4: Vzhled aplikace v Qoxxdoo

6.3.1 Navigační oblast

Rozložení navigační oblasti je stejné jako u předchozích implementací. Byla vytvořena komponenta Toolbaru `qx.ui.toolbar.ToolBar()`, do které se postupně vsazovala jednotlivá tlačítka. Tlačítka samozřejmě obsahují ikony. Připojení složky s ikonami do aktuálního projektu se vytvoří pomocí deklarace před samotnou třídou. Příklad 6.4 ukazuje možnosti deklarací.

```
// příklad deklarace pro složku s ikonami o velikostech 16x16 pixelů
#asset(qx/icon/${qx.icontheme}/16/actions/*)
```

Listing 6.4: Příklad deklarace ikon do třídy projektu

6.3.2 Adresářová struktura

Zobrazení výpisu složek je provedeno s využitím komponenty `qx.ui.tree.Tree()` kde je využito datových skladů. Pro uchování dat byl zvolen datový sklad pro JSON.

6.3.3 Výpis emailových zpráv

Pro výpis emailových zpráv je i zde použit koncept datových skladů. Je vytvořen Model `qx.ui.table.model.Simple()`, kterému jsou nastaveny popisky jednotlivých sloupců ta-

bulky. S vužitím komponenty tabulky `qx.ui.table.Table(tableModel)` byl vytvořen nový objekt tabulky, kterému byl předán datový model dle koncepce návrhu datových skladů.

Kapitola 7

Srovnání možností knihoven

V této kapitole se pokusím srovnat možnosti jednotlivých knihoven dle zvolených hledisek. Tato hlediska jsem vybral kvůli zajímavým odlišnostem, které se nacházejí mezi jednotlivými knihovnami. Srovnání je provedeno na základních knihovnách bez využití rozšiřujících modulů třetích stran, které by samozřejmě výsledky srovnání mohly ovlivnit.

7.1 Událostní model

Událostní modely všech tří knihoven patří do vyšší úrovně abstrakce oproti registraci událostí v jazyce JavaScript. Každá knihovna obsahuje základní spojení při registrací událostí. Z tabulky 7.1 je patrné, že v každé knihovně lze pomocí jednoduchých metod umožnit za-

Název knihovny	Registrace události	Zrušení události
Dojo	`dojo.connect`	`dojo.disconnect`
jQuery	`click(), focus(), keydown() ...`	`unload()`
Qooxdoo	`addListener()`	`removeListener()`

Tabulka 7.1: Jednoduché registrace událostí

registrovat a zrušit událost na příslušný objekt. Každá knihovna má svůj událostní objekt, který tyto jednoduché události spravuje. Zjišťuje například prvek, který událost vytvořil, o jakou událost se jedná a další datové složky potřebné pro vytvoření reakce na zpracování události.

Počet druhů událostí je u každé knihovny specifický. Všechny knihovny však rozšiřují klasický událostní model definovaný konsorciem W3C 3.3.2 a zvyšují tak komfort při ovládání aplikace samotné. Knihovny rozřišují základní události například o „zaostřování“, kdy je možné sledovat, zda se kurzor myši pohybuje nad konkrétním prvkem či komponentou. Dále pak např. události jako `mouseleave`, kdy je možné sledovat odjetí myši nebo události zaměřené na „rolování“ kolečka myši, které je u osobních počítačů v dnešní době velice rozšířené. Mezi zajímavé vlastnosti bych také rád zmínil metodu `Command()` knihovny Qooxdoo. Knihovna prostřednictvím této metody rozšiřuje základní podporu stisku unikátních kláves a přidává podporu klávesových zkratek. Ty je možné registrovat a vytvořit příslušné události.

Pravděpodobně nejzajímavějším konceptem, ze všech srovnávaných knihoven je koncept knihovny Dojo. Dojo přidalo do podpory metody `dojo.publish` a `dojo.subscribe`. Metody umožní programátorovi řešit v projektu časté situace, kdy je potřeba zaslat reakci

na událost jednoho objektu n dalším objektům. Tímto způsobem je tedy možné vytvořit vztah 1:N. Objekt (ve významu komponenty) tedy za pomocí metody `dojo.publish` je schopen vystavit informaci o provedení události společeně s konkrétní datovou složkou a za pomocí metody `dojo.subscribe` se může libovolný počet jiných objektů „přihlásit“ k odběru těchto událostí a vytvořit tak samostanou rekaci na konkrétní událost. Návrhový vzor je velmi často používán. Výhodou je, že objekt uveřejňující událost nemusí vědět o n dalších objektech, které na tuto událost mohou reagovat a kdo tuto událost potřebuje pro změnu svého stavu, se jednoduše přihlásí.

7.2 GUI

Porovnávání knihoven z hlediska vrstvy grafického uživatelského rozhraní (GUI) je poněkud sporné. Pokud bych měl seřadit knihovny dle výše úrovně abstrakce, je tento pohled jasný. Knihovna jQuery je z pohledu abstrakce na nejnižším místě jelikož přímo pracuje a navazuje na elementy daného html dokumentu. Proto vzhled grafického uživatelského rozhraní tvoří především vzhled definovaný programátorem pro dokument, na který navazujeme vrstvou JavaScriptu. Oproti porovnávaným knihovnám totiž nedefinuje prvky GUI jako layouty, toolbary, rozbalovací menu, stromové struktury a tlačítka. Předpokládá se, že tyto prvky si uživatel v dokumentu vytvoří sám za pomocí nižších vrstev jako HTML a CSS a na ně definuje konkrétní akce nebo využije modulárního systému a využije komponenty třetích stran. Proto zde srovnávám knihovnu jQuery v jejím rozšíření UI. V době psaní tohoto textu se knihovna nacházela ve verzi 1.7.2. Od verze 1.8 má v plánu obsahovat již základní sadu rozšířených komponent, jako tlačítko, automatické doplňování textu do vstupního pole a další potřebné věci k definování jednotného grafického rozhraní. To by mohlo být začátkem směřujícím ke komplexnější podobě GUI.

Vytváření vlastních komponent pro jQuery může být velmi náročné. Knihovna má možnost definovat množství barevných témat, které barevně sjednotí komponenty uživatelského rozhraní. U komponent použitých z knihovny UI je barevná jednotnost samozřejmá. Pokud si ale uživatel chce dodefinovat svoji konkrétní komponentu tak, aby byla barevně nastavitelná dle témat jQuery, je nucen prohledávat jednotlivé kaskádové styly použité vývojáři a ve své komponentě je uvádět.

Oproti knihovně jQuery je tomu grafické uživatelské rozhraní u knihoven Dojo a Qooxdoo o poznání lépe. Obě knihovny umožňují vzhled aplikace rozdělit za pomocí layoutů. Layouty je možné dělit na statické a flexibilní. Statické nepočítají s přepočítáváním velikosti, dle velikosti okna zatímco u dynamických je tomu naopak. Do layoutů lze pak pohodlně umísťovat jednotlivé komponenty. Tento způsob je dle mého názoru zdařilý a umožní tak vytvářet vzhledově a orientačně vhodné uživatelské rozhraní. Výhodou je i strukturovanost aplikace a dynamická možnost měnit komponenty v příslušném layoutu.

7.3 Podporované prohlížeče

Důležitým faktorem při srovnání knihoven je podpora ze strany prohlížečů. Každá knihovna podporuje jiné verze prohlížečů. Tabulka 7.2 zobrazuje dnes nejpoužívanější webové prohlížeče a jednotlivé verze, ve kterých současné verze knihoven zaručují funkčnost. Výčet podporovaných verzí prohlížečů se neustále mění, dle nových funkcí knihoven nebo vylepšení. Knihovna jQuery například do tohoto výpisu ještě přidává prohlížeče na os Linux, jako je Konqueror apod. Tabulka udává výčet verzí webových prohlížečů, ve kterých je vý-

vojáři zaručena 100% funkčnost. V nižších verzích prohlížečů knihovny sice můžou fungovat, ale jejich funkce není kompletně zajištěna se uvádí.

	JQuery UI 1.7.2	**Dojo 1.4**	**Qooxdoo 1.0.1**
Internet Explorer	6+	6+	6+
Mozilla Firefox	2+	3+	1.5+
Opera	9+	9+	9+
Chrome	1+	3	3+
Safari	3.1+	4	2+

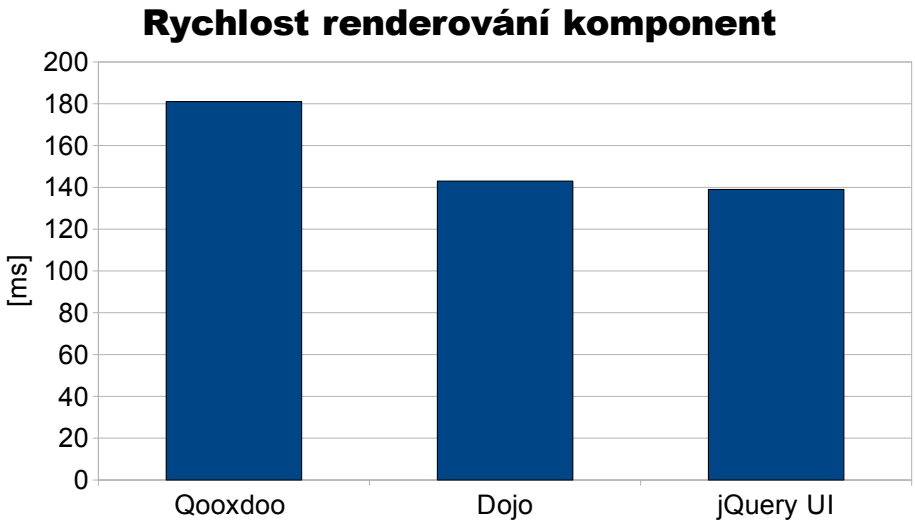
Tabulka 7.2: Porovnání jednotlivých podporovaných prohlížečů

Při porovnání jednotlivých knihoven y hlediska podporovaných prohlížečů jsem získal informace z oficiálních dokumentací jednotlivých knihoven [6], [13] a [9].

7.4 Rychlost renderování

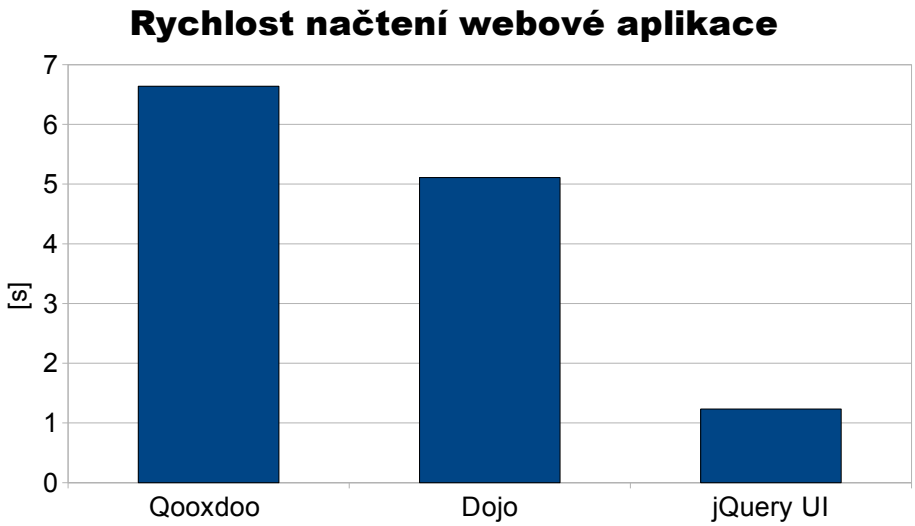
Dalším kritériem pro porovnání knihoven je porovnání z hlediska rychlosti renderování komponent v aplikaci. Pro účely testu jsem vybral zásuvný modul Speed Tracer [18] do webového prohlížeče Google Chrome 5.0.375.38. Modul byl vytvořen společností Google a umožňuje sledovat rychlosti načítání kódu v jazyce JavaScript, CSS, DOM, AJAX požadavků na server a rychlosti renderování komponent webové stránky.

S využitím modulu jsem provedl měření rychlosti renderování všech komponent a rychlosti načtení kompletní aplikace. Výsledky testů jsou zobrazeny v grafech 7.1 a 7.2.



Obrázek 7.1: Graf rychlosti renderování komponent

Z grafu je patrné, že rychlosti renderování jednotlivých komponent knihoven jsou takřka totožné. Patrně je to dáno kvalitní optimalizací knihoven o kterou se zasazují vývojáři. V případě testu rychlosti načtení celkové aplikace dopadla nejlépe knihovna jQuery. Je to dáno zejména jejím jednoduchým konceptem a tím, že defakto neobsahuje pokročilé prvky



Obrázek 7.2: Graf rychlosti načtení webové aplikace

GUI, které zásadně ovlivňují vzhled a rychlost načítání aplikace. Z výsledků je také patrná nízká rychlost načtění knihovny Qooxdoo.

Kapitola 8

Závěr

V práci jsem se zabýval možnostmi, které nabízí interaktivní webové aplikace s využitím knihoven jazyka JavaScript. Jednalo se o knihovny Dojo, jQuery UI a Qooxdoo. Detailně jsem prostudoval možnosti knihoven z hlediska koncepce, komunikace a způsobů práce s knihovnou. Poté jsem přistoupil k návrhu implementace a navrhl jsem koncept jednoduchého emailového klienta. Z hlediska klientského bylo vytvořeno uživatelské rozhraní aplikace umožňující čtení emailových zpráv s možností odeslání nové emailové zprávy. Dalšími možnostmi klienta je přeposílání, odpovězení a smazání vybrané emailové zprávy. Pro účely testování a předvedení klienta bylo navrženo serverové rozhraní napsané v jazyce PHP, které umožňuje za pomocí IMAP protokolu přístup ke konkrétní emailové schránce.

Podstatnou částí práce bylo srovnání přístupů jednotlivých knihoven a možností, které nabízí při tvorbě grafického uživatelského rozhraní. Srovnání bylo provedeno také z hlediska komponent. Dvě z knihoven (Dojo a Qooxdoo) nabízí již v samotném základu širokou škálu komponent. Třetí knihovna jQuery využívá jiné strategie, a to modulárního systému k dodatečnému využití komponent třetích stran při zachování minimálního a rychlého jádra. Kritériem pro srovnání byla také práce s daty, kde knihovny Dojo a Qooxdoo vytváří datové abstrakce ke snadnějšímu přiřazení dat ke konkrétním komponentám.

Závěrem práce jsou provedena srovnání jednotlivých knihoven z hlediska událostního modelu, grafického uživatelského rozhraní, podporovaných prohlížečů a rychlosti renderování a načtení aplikace. Srovnání knihoven bylo vytvořeno na základě získaných poznatků při implementaci vzorových aplikací, kde bylo využito maximálních možností, které knihovny nabízejí.

Mezi knihovnami jsou patrné rozdíly. Jednoznačný rozdíl lze naleznout v rozsahu, kdy knihovny Dojo a Qooxdoo jsou obsáhlejší, než knihovna jQuery. Oproti jQuery obsahují velké množství komponent, témat, barevných ikon, abstraktních typů apod. Za zmínění stojí komponenty tzv. datových skladů, kdy z různých datových zdrojů lze vytvořit objekt (datový sklad), který reprezentuje data. Propojení datových skladů s komponentou je poté velmi triviální. Knihovna jQuery oproti ostatním zahrnuje obrovské možnosti z hlediska manipulace s objektovýmu modelem dokumentu, kdy využívá možností CSS selektorů pro výběr konkrétní množiny prvků dokumentu. Z hlediska implementace GUI se jeví jako zajímavý koncept knihovny Qooxdoo. Vývoj GUI se zaměřuje pouze na komponenty a jejich umístění v layoutovacích prvcích. Při vývoji aplikace není možnost zasahovat do nižších vrstev jako HTML či CSS. Tyto vrstvy je možné měnit pouze za pomocí metod vybraných objektů.

8.1 Přínos práce

Podařilo se mi vytvořit vzorové implementace emailových klientů ve vybraných knihovnách, které lze využít jako jednoduché komunikační prostředky. Jednoznačný přínos práce vidím v zhodnocení vlastností jednotlivých knihoven, kdy volba knihovny pro implementaci daného problému je nesnadná a může být klíčová.

Ačkoliv vzorová implementace v této práci má sloužit pro porovnání knihoven, tak stále existují možnosti rozšiřitelnosti. Možnost rozšíření může spočívat v dodatečných funkcích na úroveň desktopových aplikací jako jsou Microsoft Outlook nebo Mozilla Thunderbird. Rozšíření by se mohlo týkat odesílání emailových zpráv s přílohou, filtrů, adresářových kontaktů apod.

Literatura

[1] Index of the HTML 4 Elements. 1999, [Online; navštíveno 20.01.2010].
URL http://www.w3.org/TR/html4/index/elements.html

[2] XHTML 1.0: The Extensible HyperText Markup Language (Second Edition). 2002, [Online; navštíveno 10.5.2010].
URL http://www.w3.org/TR/xhtml1/#general

[3] W3C: Document Object Model (DOM). 2004, [Online; navštíveno 21.11.2009].
URL http://www.w3.org/TR/DOM-Level-3-Core/

[4] XML Events 2. 2008, [Online; navštíveno 19.01.2010].
URL http://www.w3.org/MarkUp/2008/ED-xml-events-20081223/

[5] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. 2009, [Online; navštíveno 20.01.2010].
URL http://www.w3.org/TR/CSS2/

[6] Dojo 1.4 Release Notes. 2009, [Online; navštíveno 26.4.2010].
URL http://www.dojotoolkit.org/reference-guide/releasenotes/1.4.html

[7] Firefox Performance. 2009, [Online; navštíveno 20.1.2010].
URL http://www.mozilla.com/en-US/firefox/performance/

[8] qooxdoo — Data Binding. 2009, [Online; navštíveno 17.5.2010].
URL http://qooxdoo.org/documentation/0.8/data_binding

[9] qooxdoo — Feature Comparison. 2009, [Online; navštíveno 26.4.2010].
URL http://qooxdoo.org/documentation/general/feature_comparison

[10] qooxdoo Architecture. 2009, [Online; navštíveno 9.1.2010].
URL http://qooxdoo.org/documentation/1.0/architecture

[11] Facebook Statistics. 2010, [Online; navštíveno 10.5.2010].
URL http://www.facebook.com/press/info.php?statistics

[12] jQuery Grid Plugin. 2010, [Online; navštíveno 18.1.2010].
URL http://www.trirand.com/blog/

[13] jQuery UI — Demos & Documentation. 2010, [Online; navštíveno 26.4.2010].
URL http://jqueryui.com/demos/

[14] jQuery UI — ThemeRoller. 2010, [Online; navštíveno 10.1.2010].
URL http://jqueryui.com/themeroller/

[15] jQuery UI.Layout Plug-in. 2010, [Online; navštíveno 24.1.2010].
URL http://layout.jquery-dev.net/

[16] jsTree. 2010, [Online; navštíveno 20.1.2010].
URL http://www.jstree.com

[17] qooxdoo — Mixins. 2010, [Online; navštíveno 14.05.2010].
URL http://qooxdoo.org/documentation/1.1/mixins

[18] speedtracer. 2010, [Online; navštíveno 17.5.2010].
URL http://code.google.com/p/speedtracer/

[19] SunSpider JavaScript Benchmark. 2010, [Online; navštíveno 20.1.2010].
URL http://www2.webkit.org/perf/sunspider-0.9/sunspider.html

[20] Bernard, B.: Rich Internet Applications v roce 2008. 2008, [Online; navštíveno 20.1.2010].
URL http://qooxdoo.org/documentation/1.0/architecture

[21] Janovský, D.: Jak Psát Web. 2009, [Online; navštíveno 8.1.2010].
URL http://www.jakpsatweb.cz

[22] Jeff Croft, J. B., Dan Rubin: *Mistrovství v CSS*. Brno: Computer Press, 2007, ISBN 978-80-251-1705-7, 409 s.

[23] Pichlík, R.: Interval.cz: Rich Internet Application. 2005, [Online; navštíveno 20.01.2010].
URL http://interval.cz/clanky/rich-internet-application/

[24] Resig, J.: *Pro JavaScript Techniques: The Ultimate JavaScript book for the modern Web Developer*. Apress, 2006, ISBN 978-1-59059-727-9, 350 s.

[25] Rich Gibson, S. E.: *Google Maps hacks*. AO'Reilly Media, 2006, ISBN 978-0-596-10161-9, 337 s.

[26] Russell, M. A.: *Dojo: The Definitive Guide*. O'Reilly media, 2008, ISBN 978-0596516482, 496 s.

[27] Steven Holzner, J. Z.: *Mistrovství v AJAXu*. Brno: Computer Press, 2007, ISBN 978-80-251-1850-4, 591 s.

Dodatek A

Obsah CD

- vzorová implementace emailových klientů
- pdf dokumentace obsahující návod k instalaci, odkazy na funkční implementaci a popis serverového rozhraní
- spustitelná verze prohlížeče Google Chrome
- zdrojový text bakalářské práce ve formátu LaTeX
- text bakalářské práce ve formátu PDF